7-2017

Real-time bursty topic detection and virality forecasting in microblogs

Wei XIE
Singapore Management University, wei.xie.2012@phdis.smu.edu.sg

REAL-TIME BURSTY TOPIC DETECTION AND VIRALITY FORECASTING IN MICROBLOGS

WEI XIE

SINGAPORE MANAGEMENT UNIVERSITY

2017

Real-time Bursty Topic Detection and Virality Forecasting in Microblogs

by

Wei Xie

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Information Systems

Dissertation Committee:

Feida Zhu (Supervisor/Chair)
Assistant Professor
Singapore Management University

Jing Jiang
Associate Professor
Singapore Management University

Ee-Peng Lim
Professor
Singapore Management University

Ke Wang
Professor
Simon Fraser University

Singapore Management University
2017

Real-time Bursty Topic Detection and Virality Forecasting in Microblogs

by

Wei Xie

Abstract

Microblogs such as Twitter have become the largest social platforms for users around the world to share anything happening around them with friends and beyond. A bursty topic in microblogs is one that triggers a surge of relevant tweets within a short period of time, which often reflects important events of mass interest. How to leverage microblogs for early detection and further impact analysis of bursty topics has, therefore, become an important research problem with immense practical value.

To address the above problem, we propose in this dissertation a framework which contains the following three parts. The first part is to select a budgeted set of users as information sources for early detection of bursty topics. We first formulate this problem as a constraint satisfaction problem which has high computational complexity. To reduce the computational cost, we then transform the problem into an LP (Linear Programming) problem. Furthermore, we use the sub-gradient method instead of the standard simplex method or interior-point method to solve the LP problem, which makes it possible for our solution to scale up to large social networks. The second part is that of detecting bursty topics from a tweet stream. We propose a two-stage integrated solution TopicSketch. In the first stage, we design a small data sketch which efficiently maintains at a low computational cost the acceleration of two quantities: the occurrence of each word pair and the occurrence of each word triple. In the second stage, we propose a sketch-based topic model to infer both the bursty topics and their acceleration based on the statistics maintained in the data sketch. A dimension reduction technique based on hashing is also

proposed to achieve scalability. Once a bursty topic is detected, the third part will predict the further cascade size of this bursty topic. Based on a general time-aware cascade model, in which an appropriate hazard function is designed specifically for microblog networks, a simulation-based approach is proposed to forecast the virality of the cascade. Although each part can work separately, when putting them together, we get a comprehensive solution which is able to benefit many real world applications such as social network monitoring and viral marketing.

Contents

List of Figures

List of Tables

Acknowledgements

When I completed this dissertation, I started to review how it all began. I was introduced to Assistant Professor Feida Zhu by Professor Xueqing Gong and started working with him as a research assistant in early 2011. The nice academic environment in SMU attracted me to further join the Ph.D. program here afterward under the supervision of Feida in 2012. During my long Ph.D. journey, Feida encouraged me to stretch myself and go after the goals in both research and life sides. Here I give my special thank to him. I am also grateful to Living Analytics Research Centre (LARC) for the exchange program at Carnegie Mellon University (CMU), where I had the opportunity to work with Professor Alexander J. Smola.

During my candidature, I have had the fortune of collaborating with the following brilliant researchers: Professor Ee-Peng Lim, Associate Professor Jing Jiang, Professor Ke Wang and Assistant Professor Siyuan Liu.

I would like to thank Emmy Hoang Ai Phuong for the friendship which supported me during the tough time in this journey.

How could I forget the carefree time with Ming Gao, HanBo Dai, Cen Chen, Juan Du, Freddy and all other friends? Thank all of them for their company.

Thank the staff members of LARC and SIS for their administrative support: Fong Soon Keat, Phoebe Yeo, Desmond Yap, Seow Pei Huan and others.

The research leading to the completion of this dissertation is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.

Publications based on the Dissertation

Research papers based on the dissertation (by reverse chronological order):

1. Wei Xie, Feida Zhu, Jing Xiao, Jianzong Wang. Social Network Monitoring for Bursty Cascade Detection. (Under review in ACM Transactions on Knowledge Discovery from Data, TKDD'17). (Chapter 4)

2. Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. TopicSketch: Real-time Bursty Topic Detection from Twitter. IEEE Transactions on Knowledge and Data Engineering (TKDE'16). (Chapter 3)

3. Wei Xie, Feida Zhu, Siyuan Liu and Ke Wang. Modelling Cascades Over Time in Microblogs. 2015 IEEE International Conference on Big Data (IEEE BigData'15). (Chapter 5)

4. Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. TopicSketch: Real-time Bursty Topic Detection from Twitter. Proceedings of the 13th IEEE International Conference on Data Mining (ICDM'13). (Chapter 3)

Demo system based on the dissertation:

1. Runquan Xie, Feida Zhu, Hui Ma, Wei Xie, Chen Lin. CLEar: A Real-time Online Observatory for Bursty and Viral Events. Proceedings of the 40th International Conference on Very Large Databases.(VLDB'14)

Chapter 1

Introduction

Microblogging services nowadays have made it all too easy for everyone to share or pass a piece of information to someone else. For instance, with 320 million active users and 1 billion tweets per month[1], Twitter provides an easy, quick and reliable platform for users to share anything happening around them with friends and other followers. Another influential microblogging service is Sina Weibo, which has more than 500 million users in China (i.e. over a third of the population of China) and around 200 million monthly active users[2].

In particular, it has been observed that, in certain life-critical disasters of societal scale, microblogging services are the most important and timely sources from which people find out and track the breaking news before any mainstream media picks up on them and rebroadcast the footage. For example, on March 11, 2011, Japan earthquake and subsequent tsunami, the volume of tweets sent spiked to more than 5,000 per second when people post news about the situation along with uploads of mobile videos they had recorded [3]. We call such topics which trigger a surge of a large number of relevant tweets *bursty topics*, the entire diffusion process of a piece of message through users a *cascade* and each user who shares information with the public a *social network sensor*[4].

[1]https://about.twitter.com/company
[2]http://www.whatsonweibo.com/sinaweibo/
[3]http://blog.twitter.com/2011/06/global-pulse.html
[4]In Chapter 5, we use these two terms *sensor* and *user* interchangeably.

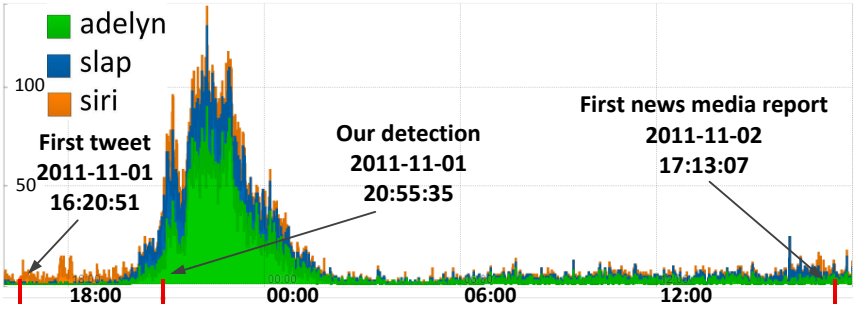


Figure 1.1: The tweet volume of each of the top three keywords of the topic: "adelyn", "slap" and "siri".

Figure 1.1 shows an example of a bursty topic on November 1st, 2011. A 14-year-old girl from Singapore named Adelyn (not her real name) caused a massive uproar online after she was unhappy with her mother's incessant nagging and resorted to physical abuse by slapping her mother twice, and boasted about her actions on Facebook with vulgarities. Within hours, it soon went viral on the Internet, trending worldwide on Twitter and was one of the top Twitter trends in Singapore. For many bursty events like this, users would like to be alerted as early as it starts to grow viral. However, it was only after almost a whole day that the first news media report on the incident came out. In general, the sheer scale of microblogs has made it impossible for traditional news media, or any other manual effort, to capture most of such bursty topics in real-time even though their reporting crew can pick up a subset of the trending ones. This gap raises a question of immense practical value: *Can we leverage microblogging services for automated bursty topic detection in real-time?*

Consequently, once a bursty topic is detected, it is natural to ask questions like: "*How many users will it eventually reach?*" With the fact that the size of a cascade often translates into the influence of a message (e.g. the social impact of a piece of breaking news), prediction of the size of a cascade, especially at the early stage of its growth, is critical in identifying among billions the truly viral messages, so that we can monitor, trace and leverage their impacts.

Besides, from the perspective of a third party, a data stream with sufficient

social network sensors is not always available for various reasons, e.g. privacy issue or commercial concerns. For instance, Sina Weibo offers no free streaming API and third party entities are subject to stringent restrictions on data collection. Although Twitter provides free streaming API, it reserves the right to charge for commercial purposes [5]. It remains an interesting question to investigate: *How to select a budgeted set of social network sensors to form the data stream for bursty cascade detection without compromising the detection performance.*

It is nearly impossible to address all the above three questions in one solution. For example, usually, we use cascade size to measure the virality of a topic. However, as shown in Section 3.2.1, it is not a good indicator for burst detection at early-stage. So in this dissertation, we propose a solution to address each of the above three questions separately. In order to solve them separately, we make the following assumptions. For bursty topic detection, we assume that a data stream with sufficient social network sensors is always available. For virality forecasting, we assume that we observe the growth of a cascade at its early-stage. For social network sensor selection, we assume that a proper burst detection approach is given. We introduce these solutions in more details as follows.

1.1 Bursty Topic Detection

In fact, the above real-time bursty event detection task has not been fully addressed by existing works on Twitter topic analysis. First, Twitter's own trending topic list does not help much as it reports mostly those all-time popular topics, instead of the bursty ones that are of our interest in this work. Second, most prior research works define a bursty topic as a set which consists of few bursty words [80, 82, 67, 13, 93, 44]. As only bursty words are captured, the represented bursty topic is far from informative to reflect what the topic

[5]https://dev.twitter.com/overview/terms/agreement-and-policy

really is. Third, most topic modelling based works study the topics in Twitter in a retrospective off-line manner, e.g., performing topic modelling, analysis and tracking for all tweets generated in a certain time period [101, 87, 23, 90]. While these findings have offered interesting insights into the topics, it is our belief that the greatest value of Twitter bursty topic detection has yet to be brought out, which is to detect the bursty topics just in time as they are taking place. This real-time task is challenging for existing algorithms because of the high computational complexity inherent in the topic models as well as the ways in which the topics are usually learnt, e.g., Gibbs Sampling [41] or variational inference [9]. The key research challenge is how to solve the following two problems in *real-time*: (I) How to efficiently maintain proper statistics to trigger detection; and (II) How to model bursty topics without the chance to examine the entire set of relevant tweets as in traditional topic modeling. While some work such as [80] indeed detects events in real-time, it requires predefined keywords for the topics.

For the above reasons, in this dissertation we propose a new detection solution – TopicSketch. It can be observed from Figure 1.1 that TopicSketch is able to detect this bursty topic within 6 hours after the very first tweet about this incident was generated, just when it started to grow viral and much earlier than the first news media report. This part of our research is covered in Chapter 3.

1.2 Virality Forecasting

There are a lot of research works which focus on predicting the cascade size, especially the size of viral one, e.g., [17] [53] [60] [48] and [6]. Despite the different approaches in their solutions, one thing in common is that all of them must have real viral cascade instances as input, the more the better. Unfortunately, however, in real life truly viral cascades are few and far in

between as compared to the whole set of tweets (in our dataset only 1‰ re-tweeting cascades grow over 35), resulting in a challenge for all existing solutions to further enhance their prediction accuracy. One solution to alleviate the rarity of viral cascades is to be able to simulate cascades that well fit the real viral ones. This motivates us to integrate the time dimension into cascade modeling for each particular cascade.

There has been several cascade models, including the well-known Threshold Model [40], Cascade Model [36] in which the time fact is not a concern, and recent works such as [37] [70] [39] [28], which focus on integrating the time dimension into the model. Particularly, in these recent works a hazard function of time is used to model how information diffuse over time in networks. In this dissertation, we call them time-aware cascade models. Following this line, we build a general time-aware cascade model for each particular cascade to depict its development over time in the social network. By applying the proposed time-aware model to real cascades in Twitter, we propose an appropriate hazard function specifically for Twitter network. We show that, by simulating more viral cascades, better virality prediction performance can be achieved. We cover this part of our research in Chapter 4.

1.3 Social Network Sensor Selection

From different perspectives, several solutions have been proposed to solve sensor selection problem in networks, e.g. maximum coverage problem [56, 84], influence maximization problem [55, 16, 15], outbreak detection problem [64, 18, 33]. Different from all these previous works, in this dissertation we first put the sensor selection problem in the circumstances of bursty cascade detection, which is not easy.

The challenges come from a few aspects: First of all, different detection approaches have been proposed in literature with different characteristics and

priorities. It is not easy to design a uniform sensor selection solution for all these different approaches. Secondly, as a classical combinational optimization problem, sensor selection problem (or sensor placement problem) has been proven to be NP-hard [56, 58]. Thirdly, although some suboptimal solutions (e.g. CELF in [64]) can significantly reduce the computational cost, it is still difficult to make them scalable to large social networks which contain millions of users, if not more. In other words, efficiency is crucial for this task.

In Chapter 5, we respond to these challenges step by step as follows.

First, by generalizing different detection solutions as an abstract classifier, we formulate the above-mentioned problem as a constraint satisfaction problem as shown in Section 5.1. Therefore we are able to design one uniform sensor selection solution for all different detection approaches, rather than "handcraft" a specific solution for each of them. Secondly, to reduce the complexity of the constraint satisfaction problem, in Section 5.2 we relax the constraints of this problem and further transform it into an LP (Linear Programming) problem which can be solved in polynomial time. Thirdly, by exploiting the special form of the above LP problem, we show in Section 5.3 that the LP problem is equivalent to a convex optimization problem. Furthermore, we propose to use the sub-gradient method to scale up the solution.

1.4 Framework

Although above each piece of work is independent of the others, when putting them together, we get a comprehensive framework (as shown in Figure 1.2) for bursty and viral topic detection. With the purpose of detecting bursty cascades, in part (I) we select a budgeted set of users. By monitoring this set of users, we get a data stream containing all the tweets generated by these users. In part (II) we monitor this stream, maintain a statistic measuring the burstiness of the stream and infer bursty topics from the statistic once

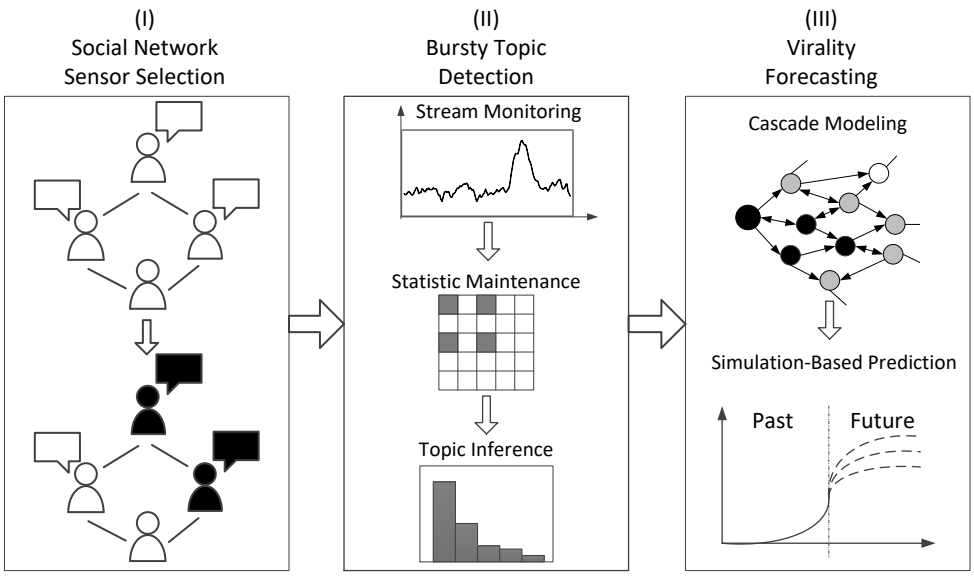


Figure 1.2: Framework.

the burstiness score exceeds a predefined threshold. In this way, most trivial cascades are filtered out in part (II), and only a small number of cascade candidates are provided with part (III). Based on a time-aware cascade model, in part (III) we predict the cascade virality by using a simulation-based approach. Because among these three parts, part (II) is crucial, we first present part (II) in Chapter 3, followed by the other two parts in Chapter 4 and Chapter 5 respectively.

1.5 Contributions

To summarize, we make the following contributions in this dissertation:

1. We propose a framework for real-time bursty topics detection and virality forecasting in microblogs. This framework consists of three separated parts: (I) social network sensor selection, (II) bursty topic detection and (III) virality forecasting. When putting these three parts together in a pipeline, it provides us with a comprehensive solution for bursty and viral topic detection.

2. For real-time bursty topics detection, we propose a two-stage integrated solution TopicSketch. In the first stage, we propose a small data sketch which efficiently maintains at a low computational cost the acceleration of two quantities: the occurrence of each word pair and the occurrence of each word triple. These accelerations provide as early as possible the indicators of a potential surge of tweet popularity. They are also designed such that the bursty topic inference would be triggered and achieved based on them. The fact that we can update these statistics efficiently and invoke the more computationally expensive topic inference part only when necessary at a later stage makes it possible to achieve real-time detection in a data stream of Twitter scale. In the second stage, we propose a sketch-based topic model to infer both the bursty topics and their acceleration based on the statistics maintained in the data sketch. Furthermore, we propose dimension reduction techniques based on hashing to achieve scalability and, at the same time, maintain topic quality with robustness.

3. For virality forecasting, we build a time-aware cascade model for each particular cascade. Furthermore, we make two key observations on user retweeting behaviour and develop four constraints based on which we design an appropriate hazard function for the model. We present an illustrative example as well as extensive experiments to demonstrate how our model fits to the real data. We would like to point out that, although hazard function has been used to model information propagation in general, in this work we make effort to design customised hazard functions based on properties of information diffusion in microblogging services. More importantly, we propose a strategy to make use of the simulations of our model to remedy the imbalance issue in cascade data. It is shown with experiments that the prediction performance improves as a result of the strategy with more viral cascades successfully identified.

4. For social network sensor selection, we propose a general sensor selection problem for different burst detection approaches. In general, sensor selection problem is NP-hard. Especially for the large social networks with millions of users, existing greedy methods are hardly scale to such size. After formulating this problem as a constraint satisfaction problem, we transform it into an LP (Linear Programming) problem which has only few constraints. Furthermore, we develop a sub-gradient algorithm to solve the LP problem, which makes it possible for our solution to scale up to large social networks. Compared with existing solutions, our solution can find better set of sensors for burst detection.

1.6 Organization of the Dissertation

The research written in this dissertation is an aggregation of several research papers we have written. In Chapter 2, we first review some research that are related to ours. Then in Chapter 3, we introduce in details our real-time bursty topic detection system TopicSketch [96, 97, 94], and compare it with other existing event detection systems. In Chapter 4, we build a general time-aware cascade model for each particular cascade, and apply it to predict viral cascades [98]. In Chapter 5 we further discuss how to select budgeted social network sensors as information sources, in order to form a data stream as the input of TopicSketch. We conclude this dissertation and discuss our future works in Chapter 6.

Chapter 2

Related Work

We review some related works that motivate our research in this dissertation. We would first review existing research on bursty topic detection — from classical event detection to latest topic modeling based approaches. We would then review existing works on social network sensor selection. We end this chapter by examining the related literature on information cascade and virality forecasting. In each section, we highlight the main differences between our dissertation and the research in existing literature.

2.1 Bursty Topic Detection

Event detection has been studied for decades, with evolving interests on news [2, 100], blogs [73] and recently social media [67, 80]. As there are numerous research works focusing on it, here we categorized the ones most related to this dissertation, i.e. bursty topic detection in microblogs, along two dimensions: the way of defining a topic and the way of processing data (as shown in Table 2.1).

Clustering Based vs. Topic Modelling Based. There are different ways to define the topic of an event. In the early work First Story Detection [100, 2] and their successors [11, 71], a topic is represented as a cluster of related documents. By exploiting the temporal proximity of news stories discussing a given

Table 2.1: The related work is categorised along two dimensions: the way of defining a topic and the way of processing data.

	Clustering Based	Topic Modelling Based
Retrospective	[100]	[101][87][23][90]
Online / Real-time	[100][2][11][71][80][82][3][65][67][13][93][44][30]	TopicSketch

event, Yang et al. [100] use refined hierarchical and online document clustering algorithms to detect events from a news stream. In [2] each document is represented as a point in a vector space (e.g. TF-IDF vector), and for each new incoming document, compare it against earlier points. If the new point is close enough to its nearest neighbour, it is absorbed by its nearest neighbour. Otherwise, this new document is labelled as a new event. Brants et al. [11] extend [2] by using incremental TF-IDF model, sophisticated similarity score normalisation etc. However, this approach does not scale to the overwhelming data volume like that of Twitter, as a nearest neighbour search is costly on large data set. Petrovic et al. [71] use locality sensitive hashing (LSH) [51] to scale this approach for Twitter streams. While in other works, a topic is defined as a coherent set (or cluster) of key words [80, 82, 67, 13, 93, 44], hashtags [3, 30], phrases [63] or segments [65]. In such works, usually a collection of bursty terms are detected from the document stream based on some criteria, and possibly later these bursty terms are grouped into several clusters which represent the bursty topics. For instance, the state-of-the-art solution SigniTrend [82] first detects the significant trending terms (words and word cooccurrences) based on its proposed statistic which measures the significances of the terms. With little memory, this statistic can be efficiently updated in an incremental way. Furthermore, by using hashing techniques, it makes it possible to track all the keyword pairs under a fixed amount of memory. Finally, an end-of-day analysis is performed, which aggregates the detected keywords into larger topics by using clustering approaches.

On the other hand, using a probability distribution over words to repre-

sent a topic has been quite common in topic modelling [46, 9]. Particularly, the words with high probabilities would characterise the topic well. There are several existing research on building topic model on document stream [8] [91][89][12][35][45][78]. It is straightforward to learn the topics in the document stream using topic models, and then find the bursty ones by considering their temporal information, such as [87, 23]. Takahashi et al. [87] first use DTM (dynamic topic model) [8] to learn the topics from news stream, and then applies Kleinberg's model [57] to detect the bursty topics. Similarly, Diao et al. [23] build a topic model which simultaneously considers both the temporal information of tweet and user's personal interests to learn the topics from tweet stream, and then Kleinberg's model is used to find the bursty ones. Other related work includes [90] which builds a topic model to find correlated bursty topics from coordinated text streams, and [101] which creates a unified model to distinguish temporal topics from stable topics. In recent works [47, 103] topic models are built to discover geographical topics from Twitter. Although the direct focuses of these works are not on bursty topics, using the similar way as above, geographical bursty topics could be found.

Besides, Ahmed et al. [1] propose a time-dependent topic-cluster model, which combines LDA [9] and clustering to learn the topic of each storyline, and at the same time, to cluster documents into storylines. And similar to [1], in latest work [26] Du et al. cluster continuous-time document streams by proposed Dirichlet-Hawkes Process. Although their model are built on general document streams, they can be potentially adopted to detect bursty topics in microblogs.

Retrospective vs. Online / Real-time. In early work [100] Yang et al. propose methods for both retrospective and online event detection. In the former case, it is assumed that there is a retrospective view of the data in its entirety. On the other hand, in the case of online event detection, the system processes current document before looking at any subsequent documents. It is

not surprising that [100] shows the results of retrospective detection are much better than the online one, as more information is available from a retrospective way. As we summarised in Table 2.1, most topic modelling based methods [101, 87, 23, 90] fall into this category. The complexity of their models makes them good at learning topics from the data in a retrospective way, but at the same time lose the flexibility to respond to any new incoming data. In contrast, methods like [82] only need to maintain a statistic for each term, and report the terms when their statistics exceed the significance level. Under such a framework, online detection is a natural choice. However, the detected topic which consists of few keywords is far less informative than the topic learned from topic modelling, which is a distribution over all the words.

Besides, real-time detection is quite similar to online detection. The subtle difference between them is that in real-time detection, time is crucial, so much so that no fixed time window for detection should be assumed. The only works we are aware of that achieve real-time detection are [80] and [82]. While [80] does detect events in real-time, it needs predefined keywords for the topic, making it inapplicable to general bursty topic detection where no prior knowledge of the topic keywords is available. By incrementally updating the statistic in an efficient way, SigniTrend [82] can detect bursty keywords in real-time, but before it aggregates keywords into larger topics, it needs to wait until the end-of-day (or a fixed time period).

In this dissertation, we aim to achieve real-time bursty topic detection from the perspective of topic modelling, which distinguishes us from most existing works in taxonomy (as shown in Table 2.1). Considering the learning power of topic modelling, we expect our method provide more informative bursty topics than other existing online detection solutions .

2.2 Social Network Sensor Selection

There are several existing works on sensor selection problem (or sensor placement problem) in networks. These works include maximum coverage problem [56, 84], influence maximization problem [24, 55, 16, 15], and so on. Among them, this dissertation is most related to the ones focusing on outbreak detection. Leveraging the so-called "friendship paradox" [29, 88], Christakis et al. propose a simple heuristic that monitors the friends of randomly selected individuals from a social network as sensors for early contagious outbreak detection [18]. Other similar works include [85, 33]. Leskovec et al. study the general problem of detecting outbreaks in networks, and apply their methodology on water and blog networks [64]. They formulate this problem as an objective function optimization problem. And it is showed that objective functions such as detection time, detection likelihood and affected population are monotone submodular set functions. As in general maximizing submodular functions is NP-hard [56, 58], they propose a greedy approach called Cost-Effective Lazy Forward (CELF) to find the approximate solution with the error bound $1-1/e$. A similar greedy algorithm is also proposed in [69]. Furthermore, Zhao et al. propose a randomized greedy method to speed up the algorithm [106]. Besides, Shao et al. propose to maximize the peak lead time to find a set of people who can be monitored, so that the outbreak of flu can be detected at lead time [83].

The biggest difference between this dissertation and previous works is that, our goal is to identify the *bursty* cascades from millions of trivial ones, rather than to detect *all* of them. For example, previous work [64] optimizes the detection time, i.e. the time passed from outbreak till detection by *one* of the selected sensors. It makes sense for monitoring water pollution or detecting specific disease. However, in social networks, detecting a new hashtag by only one sensor can not make us to identify it as a burst because there are millions of hashtags, and most of them are trivial ones. We need stronger evidences, e.g. the hashtag is detected by most of the sensors in a short period of time. This

makes it hard to design a submodular objective function as in [64] for burst detection. (In Appendix C, we will discuss more about it.) Therefore, different from previous works which apply the greedy method, we choose another way — transforming the problem into an LP problem.

On the other hand, there are several existing approaches on detecting burst. According to the way of processing data, these works can be broadly classified into two categories: *online* and *retrospective* methods. Usually an online method maintains a statistic on the fly to measure the burstiness of a sequence of tweets, posts or events. And a burst is identified if the burstiness score exceeds a predefined threshold. In [97] acceleration is defined to discover bursty topics. Similarly Schubert et al. propose a significance measure to detect emerging topics at early stage [82]. Other works such as [44, 3] also fall into this category. Instead of using numerical measure, Kleinberg models the stream (or sequence) using an infinite-state automation, in which bursts appear as state transitions [57]. In order to infer the states, this method processes data in a retrospective way. Works such as [105, 50, 87, 23, 26, 4] follow this line.

In all of these detection solutions, it is assumed that the whole data stream is available. In contrast, we assume in this dissertation that there are only budgeted social network sensors available. Under this constraint, we study how to select a set of social network sensors for burst detection. To our best knowledge, this is the first work to address this problem.

2.3 Bursty Cascade Forecasting

Information diffusion or information cascade in networks has been studied for a long time. Especially after the boom of online social networks, it gets a lot of attention from the computer science researchers, e.g. on information diffusion [66] [99]; on dynamic of viral market [62][74]; on diffusion of innovations

[76][59]. We summarize the works which are most related to ours as follows.

Cascade Modelling The most influential cascade models are Threshold Model[40], Cascade Model [36] and tons of their extensions (e.g. [55], [52], [79], [61]). In the original Threshold Model and Cascade Model, time is not a factor of concern. In recent years, many works have integrated the time dimension into the cascade model for different purposes. We call them time-aware cascade models. Myers et al. model how information diffuse in networks when external out-of-network sources exist [70]. Yu et al. propose a Networked Weibull Regression model for cascade modeling [102]. Based on proposed survival model, Gomez-Rodriguez et al. study how to infer the unobserved networks [38, 37, 39]. Furthermore, Du et al. study how to uncover topic-sensitive information diffusion networks[28]. Du et al. also study scalable methods for influence estimation in diffusion networks [27]. What is common in these works is that a hazard function of time is used to model how information diffuse over time in networks. We follow this line – using a general time-aware cascade model to describe how each particular cascade grows over time. However, different from these existing works, we focus on the possible choices of hazard functions in the setting of Twitter and make effort to fit model to real cascade data. Particularly based on our observations on user retweeting behaviour, we design a specific hazard function for Twitter network.

Cascade Prediction There is also a branch of works on cascade prediction, including cascade size prediction (e.g. [48], [60], [104], [17]), and viral (or outbreaking) cascade prediction (e.g. [53], [20]). In most of these works, the prediction is based on the features learned from the training cascades. Due to lack of data at the early stage, our model is not suitable to do the cascade prediction task directly. However, we improve the prediction performance in an "orthogonal" direction – making use of the simulations of our model to remedy the imbalanced cascade data, which can potentially benefit these feature based prediction solutions.

Others Works such as [77] presents the differences in the mechanics of information diffusion across topics, [34] gives an empirical study on the structure of online diffusion networks, [25] studies characteristics of large Facebook cascades, [31] empirically studies rumour cascades on Facebook. These works provide us all kinds of insights about how cascades develop in networks. In this dissertation, we focus on the effect of time.

Chapter 3

TopicSketch: Real-time Bursty Topic Detection from Twitter

In this chapter, we present a real-time bursty topic detection solution called TopicSketch. We first give an overview of the solution. Then we propose our sketch-based topic model, which makes it possible for us to efficiently infer bursty topics just from a "snapshot" of current tweet stream. Furthermore, we present the technique details to scale up our solution to the huge volume tweet stream setting.

3.1 Overview

3.1.1 Problem Formulation

A topic in this dissertation is represented as a distribution over words. Particularly, in defining a bursty topic, we evaluate the following two criteria: (I) There has to be a sudden surge in the topic's popularity which is measured by the total number of relevant tweets. Those all-time popular topics therefore would not count; (II) The topic must be reasonably popular. This would filter away the large number of trivial topics which, despite the spikes in their popularity, are considered as noises because of the negligible number of relevant

tweets.

For criterion (I), we measure how bursty a topic is by the *acceleration* of its popularity. Mathematically speaking, acceleration captures the change in the rate of the popularity of a topic. The more sudden the change is, the larger the acceleration is. In Section 3.2.2, we explain how to estimate the acceleration of a topic without even knowing which tweets are associated to it. For criterion (II), once we found bursty topic candidates, we simply count the relevant tweets of them, and filter out the trivial ones.

Our task in this chapter is, given a tweet stream, to detect bursty topics from it as early as possible.

3.1.2 Solution Overview

Our solution, called TopicSketch, is based on two main techniques — a sketch-based topic model and a hashing-based dimension reduction technique. Our sketch-based topic model provides an integrated two-step solution. In the first step, it maintains as a sketch of the data the acceleration of two quantities: (1) every pair of words, and (2) every triple of words, which are early indicators of popularity surge and can be updated efficiently at a low cost, making early detection possible. In the second step, based on the data sketch, it learns the bursty topics by tensor decomposition. To perform the detection efficiently in large-scale real-time setting, we propose a dimension reduction technique based on hashing which provides a scalable solution to the original problem without compromising much the quality of the topics.

Figure 3.1 gives the overview of our proposed TopicSketch framework. The real-time detection flow is as follows: (1) Upon the arrival of each tweet, the sketch is updated, which is an efficient step as detailed in Section 3.3.2, (2) Once the sketch is updated, the change is sent to the monitor, (3) The monitor tracks the data sketch, compares it with previous average, and triggers the estimator for potential bursty topic detection if the difference is larger than

pre-determined threshold. (4) Upon notification, the estimator takes a snapshot of the sketch and infers the bursty topics as described in Section 3.2.3 and 3.3.3, and (5) The inferred bursty topics are sent to the reporter to evaluate and report. TopicSketch is designed such that steps (1) to (3) are computationally cheap to enable real-time response and early detection. Step (4), which is expensive if done naively, is greatly expedited by dimension reduction techniques as described in Section 3.3.1.

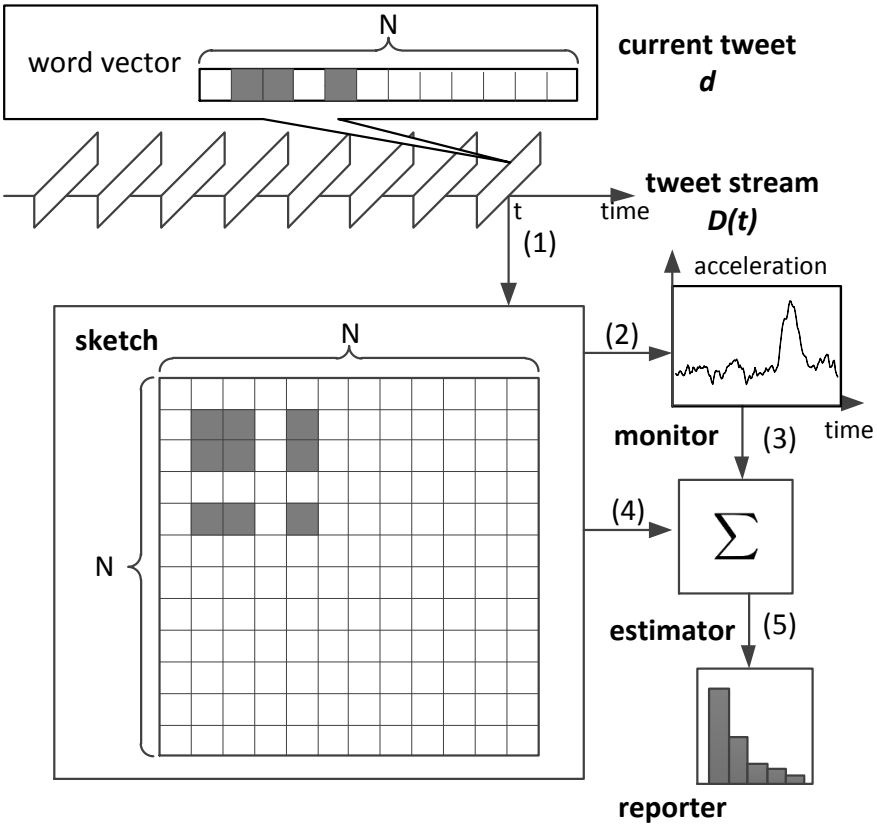


Figure 3.1: TopicSketch framework overview

3.2 Sketch-based Topic Model

3.2.1 Intuition

Actually, as mentioned in [44], the term "bursty topic" is very ambiguous, and can be viewed in very different ways. Various intuitions and corresponding definitions on it lead to diverse solutions [57, 44, 3, 82]. The intuition behind

this work comes from the observation that, the whole tweet stream is full of large amount of tweets about general topics such as car, music and food. Although they take a large proportion in the whole tweet stream, they are not helpful for our bursty topic detection task. Therefore, we try to separate the bursty topics from them. We found that, following daily routine, people usually tweet about general topics in a steady pace. In contrast, bursty topics are often triggered by some events such as some breaking news or a compelling basketball game, which get a lot of attention from people, and "force" people to tweet about them intensely. In physics, this "force" can be expressed by *"acceleration"*, which in our setting describes the change of *"velocity"*, i.e. arriving rate of tweets. Bursty topics can get significant acceleration when they are bursting, while the general topics usually get nearly zero acceleration. So the "*acceleration*" trick can be used to preserve the information of bursty topics but filter out the others. However, as the topics are hidden, we can not calculate their accelerations directly. A possible way is to estimate them by calculating the accelerations of words instead. Equation 3.1 shows how we calculate the *"velocity"* $\hat{v}(t)$ and *"acceleration"* $\hat{a}(t)$ of words[1].

$$\begin{aligned} \hat{v}_{\Delta T}(t) &= \sum_{t_i \leq t} X_i \cdot \frac{exp((t_i - t)/\Delta T)}{\Delta T} \\ \hat{a}(t) &= \frac{\hat{v}_{\Delta T_2}(t) - \hat{v}_{\Delta T_1}(t)}{\Delta T_1 - \Delta T_2} \end{aligned} \tag{3.1}$$

In Equation 3.1, X_i is the frequency of a word (or a pair of words, or a triple of words) in the *i-th* tweet, t_i is its timestamp. The exponential part in $\hat{v}_{\Delta T}(t)$ works like a soft moving window, which gives the recent terms high weight, but gives low weight to the ones far away, and the smoothing parameter ΔT is the window size. To capture the change of velocity, acceleration $\hat{a}(t)$ is defined as the difference of velocities with different window size ΔT_1 and ΔT_2. (Similar

[1]The details about how to derive this equation and other choices of smoothing are presented in Appendix A.

to the divergence of 5 day average and 10 day average in stock market, which is used to estimate the stock trend.)

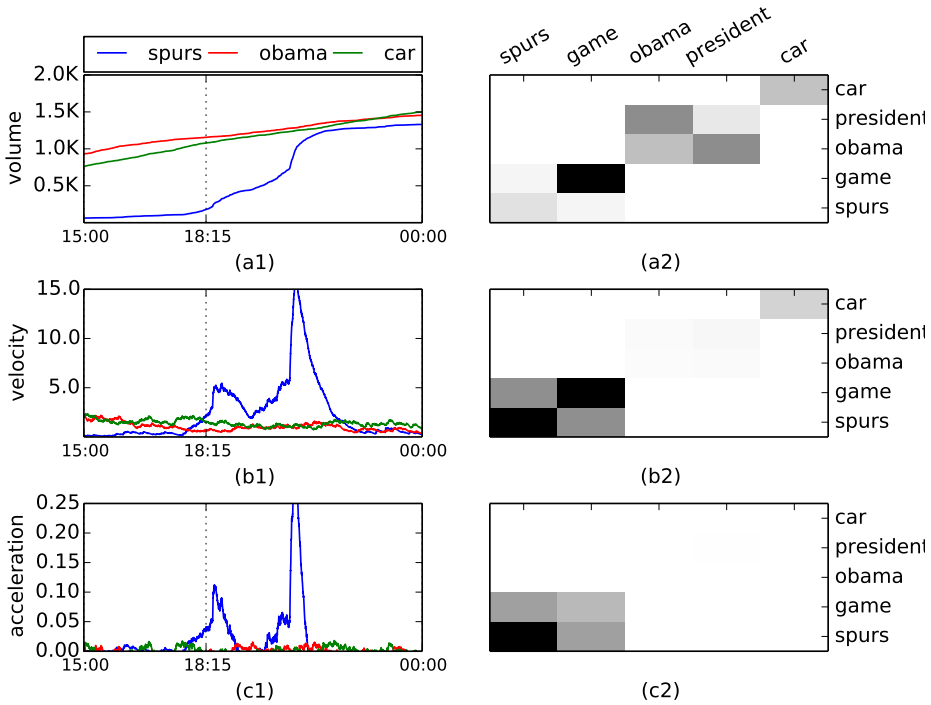


Figure 3.2: (a1), (b1) and (c1) show the daily volume, velocity and acceleration of different keywords over time respectively; (a2), (b2) and (c2) show the heat maps of these statistics on word pairs at time point 18:15.

We use real data to demonstrate the above intuition in Figure 3.2. (a1), (b1) and (c1) respectively present the daily volume, velocity and acceleration of three keywords over time, each of which represents a topic. At that day, there was a compelling basketball game between San Antonio Spurs and Oklahoma City Thunder. At the beginning, this event got a big surge in Twitter, and at the end it got another even bigger wave of discussion on this Western Conference final. The daily volume in (a1) shows the popularity of each topic. We can see that, till at the end of the day, "spurs" reaches the same scale as "obama" and "car". However, it will be too late if we wait till we observe the surge in volume to report this bursty topic. As shown in (b1), an earlier indicator is velocity, i.e. the arriving rate of a topic. Our idea of early detection is to monitor the acceleration of a topic which, compared against volume and velocity, gives an even earlier indicator of the popularity surge.

The dash line in the plot shows the time when our detection system could be triggered. It is clear that at this time point, the daily volume in (a1) and the velocity in (b1) of "spurs" are not significant enough for identifying this event. In contrast, as "obama" and "car" nearly get zero acceleration in (c1), it is easy to distinguish "spurs" from them under the measurement of acceleration. If we maintain these statistics (i.e. volume, velocity and acceleration) on word pairs instead of single words (use the same formula in Equation 3.1), we will get three dynamic matrices over time. (a2), (b2) and (c2) show the heat maps of these matrices at the detection time point respectively. In (a2), the information of all the topics are kept; in (b2) more information are kept for higher velocity topics; and in (c2) only the information of bursty topics are kept, while others are filtered out.

3.2.2 Sketch-based Topic Model

Denote $D = \{d_i\}$ as the set of all tweets generated in the tweet stream, where d_i is a tweet in the stream D, and t_i is its timestamp. Also denote $C_{i,w}$ as the number of appearance of word w in tweet d_i, and C_i as the total number of words in tweet d_i. $w \in [N]$, where N is the number of distinct words in the vocabulary.

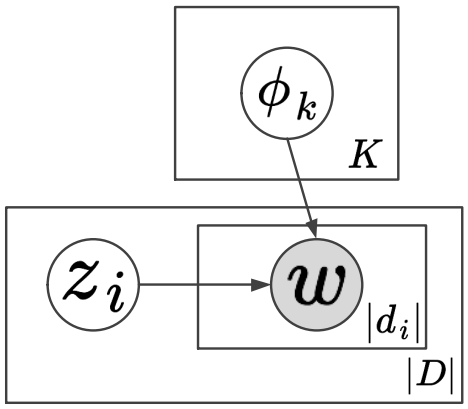


Figure 3.3: Single topic model.

We consider the following single topic model (shown in Figure 3.3). There

are K topics $\{\phi_k\}_{k=1}^K$, where each topic ϕ_k is a distribution over words. Here the key assumption is each tweet d_i is only associated to one latent topic z_i. (This assumption is based on the fact that the length of each tweet is very short, limited by 140 characters.) And each word in d_i is drawn from $Multinomial(\phi_{z_i})$.

Assume topics $\{\phi_k\}_{k=1}^K$ and topic indicator z_i are unknown but fixed, for the frequency of a word w in d_i, denoted as $f_{1,i}[w] = \frac{C_{i,w}}{C_i}$, we have

$$\mathbb{E}[f_{1,i}[w]] = \phi_{z_i}[w]$$

It means the single word frequency $f_{1,i}$ reflects the topic ϕ_{z_i}. However, as the topic indicator z_i is unknown, we can not directly infer topic from it. As shown in Figure 3.2, word pairs are useful to group different words into topics, such as the "spurs"-"game" block in the heat maps. We consider calculating the frequency of each word pair. Particularly, with the exchangeability of the words in tweet d_i (based on the single topic model above), we define the frequency of a word pair (w_1, w_2) in a tweet d_i as follows,

$$f_{2,i}[w_1, w_2] = \begin{cases} \frac{P(C_{i,w_1},2)}{P(C_i,2)} & , \quad w_1 = w_2 \\ \frac{C_{i,w_1} C_{i,w_2}}{P(C_i,2)} & , \quad w_1 \neq w_2 \end{cases} \tag{3.2}$$

where $P(C, 2)$ counts the 2-permutations of C. The denominator is the number of all possible cases choosing 2 out of C_i words in tweet d_i, while the numerator is the number of cases choosing 2 specific words. Notice that $\sum_{w_2 \in [N]} f_{2,i}[w_1, w_2] = f_{1,i}[w_1]$, which means $f_{2,i}$ actually keeps all the information in $f_{1,i}$. As the words in d_i are drawn from the same distribution $Multinomial(\phi_{z_i})$, it can be proven that

$$\mathbb{E}[f_{2,i}[w_1, w_2]] = \phi_{z_i}[w_1] \cdot \phi_{z_i}[w_2]$$

For simplicity's sake, we adopt the notation of tensor product $\otimes$[2], and denote $f_{2,i}$ as a matrix in which the $[w_1, w_2]$ element is $f_{2,i}[w_1, w_2]$. So we have an equivalent equation.

$$\mathbb{E}[f_{2,i}] = \phi_{z_i} \otimes \phi_{z_i}$$

As shown in the intuition part 3.2.1, we calculate the acceleration of word pairs to preserve most of the information for bursty topics, and at the same time filter other general topics out. Set $X_i = f_{2,i}[w_1, w_2]$ and apply Equation 3.1, we can calculate the acceleration of any word pair (w_1, w_2). Notice that the acceleration $\hat{a}(t)$ in Equation 3.1 is indeed a weighted sum of sequence $\{X_i\}$, and the weight only depends on $\{t_i\}$. In other words, $\hat{a}(t)$ can be defined as a linear function $\mathcal{A}_t(\cdot)$ on $\{X_i\}$. **Explicitly we denote $\mathcal{A}_t(\{X_i\})$ as the acceleration $\hat{a}(t)$ on $\{X_i\}$.**

For each tweet d_i, we calculate the word pair frequency matrix $f_{2,i}$, so that we have a sequence of matrices, i.e. $\{f_{2,i}\}$. Based on $\{f_{2,i}\}$, we calculate the acceleration of each word pair frequency, i.e. $\mathcal{A}_t(\{f_{2,i}[w_1, w_2]\})$. So that we have a matrix $\mathcal{A}_t(\{f_{2,i}\})$ in which the $[w_1, w_2]$ element is $\mathcal{A}_t(\{f_{2,i}[w_1, w_2]\})$. Figure 3.2 (c2) illustrates what the matrix $\mathcal{A}_t(\{f_{2,i}\})$ looks like. From the linearity of expectation, we have

$$\begin{aligned}
\mathbb{E}[\mathcal{A}_t(\{f_{2,i}\})] &= \mathcal{A}_t(\{\mathbb{E}[f_{2,i}]\}) \\
&= \mathcal{A}_t(\{\phi_{z_i} \otimes \phi_{z_i}\}) \\
&= \mathcal{A}_t(\bigcup_k \{\mathbf{1}_k(z_i) \cdot \phi_k \otimes \phi_k\}) \\
&= \sum_k \mathcal{A}_t(\{\mathbf{1}_k(z_i)\}) \cdot \phi_k \otimes \phi_k
\end{aligned} \tag{3.3}$$

where $\mathbf{1}_k(\cdot)$ is the indicator function such that if $z_i = k$, $\mathbf{1}_k(z_i) = 1$, otherwise, $\mathbf{1}_k(z_i) = 0$. On the left side of Equation 3.3, $\mathcal{A}_t(\{f_{2,i}\})$ is observable;

[2]For vectors v_1, v_2, $v_3 \in \mathbb{R}^N$, the tensor product of v_1 and v_2, $m = v_1 \otimes v_2$, is defined as a $N \times N$ matrix, where $m[i, j] = v_1[i]v_2[j]$, $i, j \in [N]$. And the tensor product of v_1, v_2 and v_3, $m = v_1 \otimes v_2 \otimes v_3$, is defined as a $N \times N \times N$ matrix, where $m[i, j, k] = v_1[i]v_2[j]v_3[k]$, $i, j, k \in [N]$.

while on the right side, $\mathcal{A}_t(\{\mathbf{1}_k(z_i)\})$ is in fact the acceleration of tweets about topic k. Thus Equation 3.3 gives us a way to link the observable acceleration of word pair frequency to the unknown acceleration of topic k without even knowing which tweets are associated to it. More importantly, Equation 3.3 actually implies that by maintaining the acceleration of word pair frequency, i.e. $\mathcal{A}_t(\{f_{2,i}\})$, we can preserve the information of the topics with high accelerations (i.e. bursty topics), and at the same time filter out the topics with nearly zero acceleration (i.e. stable general topics). That is exactly what we want.

If for any two topics $k_1 \neq k_2$, ϕ_{k_1} and ϕ_{k_2} are orthogonal, i.e. $\langle \phi_{k_1}, \phi_{k_2} \rangle = 0$, all the topics $\{\phi_k\}$ will be the eigenvectors of $\mathcal{A}_t(\{f_{2,i}\})$. If so, we can just perform one single SVD (Singular Value Decomposition) on $\mathcal{A}_t(\{f_{2,i}\})$ to infer these topics. However, $\langle \phi_{k_1}, \phi_{k_2} \rangle = 0$ means topics k_1 and k_2 have no any common words, which is far from reality. Thus we consider even higher order information, i.e. the frequency of each word triple. Similar to $f_{2,i}$, the frequency of a word triple (w_1, w_2, w_3) in a tweet d_i is defined as follows,

$$f_{3,i}[w_1, w_2, w_3] = \begin{cases} \frac{P(C_{i,w_1},3)}{P(C_i,3)} & ,w_1 = w_2 = w_3 \\ \frac{P(C_{i,w_1},2)C_{i,w_3}}{P(C_i,3)} & ,w_1 = w_2 \neq w_3 \\ \frac{P(C_{i,w_2},2)C_{i,w_1}}{P(C_i,3)} & ,w_2 = w_3 \neq w_1 \\ \frac{P(C_{i,w_3},2)C_{i,w_2}}{P(C_i,3)} & ,w_3 = w_1 \neq w_2 \\ \frac{C_{i,w_1}C_{i,w_2}C_{i,w_3}}{P(C_i,3)} & ,otherwise \end{cases} \tag{3.4}$$

Similarly, it can be proven that

$$\mathbb{E}[f_{3,i}] = \phi_{z_i} \otimes \phi_{z_i} \otimes \phi_{z_i}$$

and

$$\mathbb{E}[\mathcal{A}_t(\{f_{3,i}\})] = \sum_k \mathcal{A}_t(\{\mathbf{1}_k(z_i)\}) \cdot \phi_k \otimes \phi_k \otimes \phi_k \tag{3.5}$$

Based on Equation 3.3 and 3.5, we transform our problem into a standard

tensor decomposition problem [5], i.e. given sketches $M_2 = \mathcal{A}_t(\{f_{2,i}\})$ and $M_3 = \mathcal{A}_t(\{f_{3,i}\})$, infer topics $\{\phi_k\}$. In Section 3.2.3 we present the tensor decomposition algorithm in detail.

So by now, we have two sketches: M_2 and M_3. And each cell in these two matrices is an acceleration. For instance, the (w_1, w_2) cell in M_2, i.e. $M_2[w_1, w_2]$, is $\mathcal{A}_t(\{f_{2,i}[w_1, w_2]\})$. Notice that all these accelerations are easy to compute and update upon the arrival of every tweet (as shown in Section 3.3), which is critical for scalability in real-time setting.

As M_3 could be huge (a $N \times N \times N$ matrix), we do not really store the M_3, but project it to a $N \times N$ matrix $M_3(\eta) = \sum_w M_3[:, :, w] \cdot \eta[w]$, where $\eta \in \mathbb{R}^N$, is a random vector. However a $N \times N$ matrix is still huge, in Section 3.3, we discuss more about reducing the space complexity.

3.2.3 Topic from Sketch

To identify bursty topics from the data sketch, which consists of two matrices M_2 and $M_3(\eta)$, we employ a tensor decomposition algorithm in [5]. This algorithm first performs a SVD on M_2 to find a whitening matrix W. Afterwards, whiten $M_3(\eta)$, then perform another SVD on whitened $M_3(\eta)$ to find its eigenvectors, from which the topic vectors can be recovered. The detail is presented in Appendix B. Here we give the procedure in Algorithm 1. It has three parts: (I) Whitening, which transform $M_3(\eta)$ from a $N \times N$ matrix to a $K \times K$ matrix T_3; (II) SVD, which gets the generalized vectors $\{v_k\}$ of T_3; (III) Reconstruction, which recovers the topics $\{\phi_k\}$ and their corresponding accelerations $\{a_k\}$. As $K \ll N$, the time consuming part here is part (I), which takes time in the order of $O(K \cdot N^2)$.

Algorithm 1: TensorDecompose

Input: K: the number of topics.
M_2: the second order tensor power.
$M_3(\eta)$: the reduced third order tensor power.
Output: topics $\{\phi_k\}$ and their corresponding accelerations $\{a_k\}$.

/*Whitening */
/* $eigs(M_2, K)$ returns the largest K largest magnitude eigenvalues and corresponding eigenvectors. */
$(U, \Lambda) = eigs(M_2, K)$;
$W = U\Lambda^{-\frac{1}{2}}$; /* W may be a complex matrix */
$T_3 = W^\top M_3(\eta) W$
/* SVD */
Compute generalised vectors $\{v_k\}$ of T_3;
/* Reconstruction */
for $k = 1$ *to* K **do**
  $\phi_k = \frac{W(W^\top W)^{-1} v_k}{1_N^\top W(W^\top W)^{-1} v_k}$;
  $a_k = \frac{1}{(W^\top \phi_k)^\top (W^\top \phi_k)}$;
end
return $\{\phi_k\}, \{a_k\}$.

3.3 Realtime Detection Techniques

In this section, we present the technique details to achieve real-time efficiency for bursty topic detection in the huge-volume tweet stream setting.

3.3.1 Dimension Reduction

The first challenge is the high dimension problem as a result of the huge number of distinct words N in the tweet stream, which could easily reach the order of millions or even larger (see the experiments in Section 3.4.1). What's more, user-generated new words or hashtags always appear in Twitter. This results not only in an enormous data sketch (recall M_2 and $M_3(\eta)$ in the sketch are $N \times N$ matrices) but also a very high dimension input to Algorithm 1.

Since the problem is mainly because N is too large, one natural solution is to keep only a set of active words encountered recently, e.g. in the last 15 minutes. When a burst is triggered, consider only the words in this recent set. However, it turns out that the size of this reduced active word set for tweet

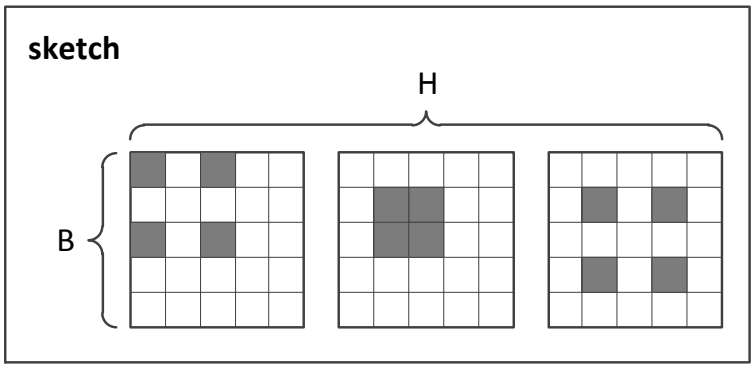


Figure 3.4: New sketch after dimension reduction.

stream is still too large (see Section 3.4.1) to infer the topics efficiently.

To handle large number of words, another common way is hashing [82]. We hash these distinct words into B buckets, where B is a number much smaller than N, and treating all the words in a bucket as one "word". Consequently, the size of the sketch becomes $O(B^2)$, which are significantly smaller than $O(N^2)$ as in the original problem. However, after hashing, what we obtain is the distribution over buckets, rather than the distribution over words. It means we would need to recover the probabilities of words from the probabilities of buckets. To solve this problem, we adapt the Count-Min algorithm [19, 54] to our setting, by using H hashing functions instead of one. In particular, it works as follows. Given H hash functions $(\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_H)$ which map words to buckets $[1...B]$ uniformly and independently. For a topic k with word distribution ϕ_k, we first estimate its distribution over buckets $\{\phi_k^{(h)}[j] = \sum_{i|\mathcal{H}_h(i)=j} \phi_k[i]\}_{j=1}^{B}$ for all the hash functions. Then we recover the probability of each word i as $\min_{1\leq h\leq H}\{\phi_k^{(h)}[\mathcal{H}_h(i)]\}$.

The sketch after hashing is illustrated in Figure 3.4. As shown in Figure 3.4, for each arriving tweet, H different bucket-pair frequencies are calculated, and H matrices are updated correspondingly. After the dimension reduction, the memory cost for the sketch is $O(H \cdot B^2)$, and the time complexity for tensor decomposition is $O(H \cdot B^2 \cdot K)$, which are small enough to be practically feasible.

We also maintain a pool of active words, so that we estimate the proba-

bility of words only in this set rather than all the words in whole vocabulary. The procedure is presented in Algorithm 2. This algorithm will estimate the probability of each word with error no greater than $\frac{e}{B}$ with a probability of e^{-N/e^H}. The details of the proof can be found in [54].

Algorithm 2: TopicRecover

Data: $active\_words$: the pool of active words.
$\{\mathcal{H}_h\}_{h=1}^{H}$: H hash functions.
$threshold$: predefined threshold.
Input: $\{\phi_k^{(h)}\}_{h=1}^{H}$: H distributions over $[B]$.
Output: $topic_k$: a topic which is represented as a set of words.

```
initialise topic_k as a empty set.;
for each word w in active_words do
    if min_{1≤h≤H}{φ_k^(h)[H_h(w)]} ≥ threshold then
        add word to topic_k;
    end
end
return topic_k
```

3.3.2 Efficient Sketch Maintenance

The core part of sketch maintenance in our solution is acceleration calculation. As presented in Section 3.2.1, we adopt Equation 3.1 to calculate acceleration. However, directly applying Equation 3.1 is far from efficient. Observed that the velocity $\hat{v}(t)$ in Equation 3.1 can be calculated in an incremental way as in the following Equation 3.6

$$\hat{v}_{\Delta T}(t) = \begin{cases} \hat{v}_{\Delta T}(t_{i-1}) \cdot e^{\frac{(t_{i-1}-t)}{\Delta T}}, & t \in (t_{i-1}, t_i) \\ \hat{v}_{\Delta T}(t_{i-1}) \cdot e^{\frac{(t_{i-1}-t)}{\Delta T}} + \frac{X_i}{\Delta T}, & t = t_i \end{cases} \tag{3.6}$$

where t_i is the timestamp of the i-th tweet. So instead of directly storing one acceleration, we incrementally maintain two velocities $\hat{v}_{\Delta T_1}(t)$ and $\hat{v}_{\Delta T_2}(t)$, from which the acceleration can be derived on the fly. Both the space complexity for maintaining one acceleration and the time complexity for updating one acceleration are therefore $O(1)$.

Another issue is that, there are $O(N^2)$ accelerations to be updated when each tweet arrives, and $O(H \cdot B^2)$ for the dimension reduced case. We take lazy maintenance strategy to reduce the computing cost. In fact, for each acceleration, take the (w_1, w_2) cell in M_2 as an example, we store a velocity pair $(\hat{v}_{\Delta T_1}(t), \hat{v}_{\Delta T_2}(t))$ and a timestamp $t^\star$ representing the last modification time. When a tweet d_i arrives, if $C_{i,w_1} \cdot C_{i,w_2} > 0$, we update the pair $(\hat{v}_{\Delta T_1}(t), \hat{v}_{\Delta T_2}(t))$ according to Equation 3.6 and update timestamp $t^\star$ to t_i, otherwise we adopt lazy strategy and simply do nothing. When a bursty topic is triggered at time t, take a snapshot of the sketch (copy on write), then recover all the velocity pairs up to time t by deriving the accelerations from the velocity pairs according to Equation 3.6. Figure 3.1 illustrates this updating procedure: When a tweet arrives, its word vector is shown. The gray cells in the word vector denotes the occurrence of relevant words in the tweet. In this example, three words occur in the current tweet. The gray cells in sketch represents the updated elements in the sketch. $O(|d|^2)$ cells are updated in total. Figure 3.4 illustrates this updating procedure for dimension reduced case. After hashing, three words in the current tweet are mapped into two buckets for three hash functions respectively. In total, $O(H \cdot |d|^2)$ cells are updated.

3.3.3 Topic Inference

Once our system is triggered, we take a snapshot of the sketch, and then infer the bursty topics from the sketch as shown in Section 3.2.3. As discussed in Section 3.3.1, instead of directly maintaining a sketch with size of $N \times N$, we maintain H sketches with size of $B \times B$, where B is much smaller than N. Here we show how to infer the bursty topics from the sketch after dimension reduction. First, we get topics from the H different sketches by *TensorDecompose* (Algorithm 1). Note this step can be implemented in parallel. One subtle problem here is that a bursty topic may have different topic indexes in different sketches (recall we have H different sketches now). For instance, sketch $h = 1$

Algorithm 3: BurstyTopicsDiscover

Data: $active\_words$: the pool of active words.
$\{\mathcal{H}_h\}_{h=1}^{H}$: H hash functions.
$threshold$: predefined threshold.
Input: K: the number of topics
Input: $\{M_2^{(h)}\}_{h=1}^{H}\{M_3(\eta)^{(h)}\}_{h=1}^{H}$: H sketches.
Output: $\{topic_k\}$: a list of topics.

```
1  /*get topics from H different sketches*/;
2  for h = 1 to H do
3  |  /* in parallel */;
4  |  {φ_k^(h)}_{k=1}^K, {a_k^(h)}_{k=1}^K = TensorDecompose(K, M_2^(h), M_3(η)^(h))
5  end
6  /* aligning topics*/;
7  for h = 1 to H do
8  |  sort {φ_k^(h)}_{k=1}^K by their corresponding {a_k^(h)}_{k=1}^K;
9  end
10 /* topic discovering */;
11 for k = 1 to K do
12 |  topic_k = TopicRecover({φ_k^(h)}_{h=1}^H);
13 end
14 return {topic_k}
```

may capture a bursty topic in topic $\phi_1^{(1)}$ with index 1, however sketch $h = 2$ may capture the same bursty topic in topic $\phi_3^{(2)}$ with index 3. To align the topics which represent the same bursty topic, we sort these topics $\{\phi_k^{(h)}\}_{k=1}^{K}$ by their corresponding accelerations $\{a_k^{(h)}\}_{k=1}^{K}$, and then re-index them according to the order. At last, we get the topics recovered by *TopicRecover* (Algorithm 2). An overview of this procedure is given in Algorithm 3. For the purpose of robust, we also propose a variant of this algorithm by using the second minimum value instead of the minimum value in Line 3 in Algorithm 2. We will discuss this more in the experiment part.

After getting the topics from Algorithm 3, we would like to add a heuristic step to refine the topics, due to the following reasons. First, because of the original acceleration based design, our solution is fragile when facing spam accounts. They usually inject a lot of duplicate or similar tweets very intensively in a short period of time, which would produce words with significant acceleration, and therefore trigger our system. Second, due to hashing collision, it

is possible that some rare words appear in the recovered topic. The refining process is as follows.

- Trivial topic filtering: filter out the topics which has less number of relevant tweets than a predefined threshold.
- Noisy topic filtering: filter out the topics which has higher entropy than a predefined threshold. Such topics look like noises, in which there are no obvious key words.
- Spam filtering: check the related tweets of a topic in recent 5 minutes. If there is an account which posts a number of tweets more than a predefined threshold, filter out this topic.
- Rare word pruning: prune the words which do not appear in the tweets of a predefined time window.

An inverted index of recent tweets is implemented, so that this step can be performed efficiently.

3.4 Evaluation

In this section, we present the evaluation of our TopicSketch system for both efficiency and effectiveness. We use two different Twitter data sets: one consists of tweets from Singapore, the other consists of tweets from San Francisco. These tweets are crawled from the Twitter users whose profile locations are Singapore and San Francisco respectively. The Singapore based data set contains 32,479,134 tweets, and the San Francisco based data set contains 99,586,724 tweets. These tweets are used to simulate live tweet streams. We implemented our prototype system in Python 2.7[3] using 64-bit addressing, and executed on multiple cores of an Intel Xeon 3.06 GHz machine.

[3]For the reason of efficiency, some core part such as sketch maintenance was implemented in C++.

To demonstrate how TopicSketch detects bursty topics on the fly, a demo is presented at `http://topicsketch.appspot.com/` for exploration (best use Chrome browser). Besides, we also have a demo paper which is based on TopicSketch [94].

3.4.1 Efficiency Evaluation

In this section, we evaluate the performance of the sketch maintenance by the throughput on the tweet stream. We also evaluate the performance of the estimator by the topic inference time. In our tweet set, after removing stop words, the average number of words in a tweets is 8 (evidence for small $|d|$). The total number of the distinct words is 8,470,180 (evidence for high dimensions). The number of distinct words in the 15-minutes active word set is between 10,000 and 20,000.

3.4.1.1 Sketch Maintenance

Before each incoming tweet goes to the component of sketch maintenance, we conduct standard preprocessing on it, including tokenizing[4], stemming, and stop words removing. After preprocessing, calculate word pair frequency $f_{2,i}$ and word triple frequency $f_{3,i}$ as shown in Equation 3.2 and 3.4. Then update the accelerations in sketch according to Equation 3.6 and our lazy maintenance strategy in Section 3.3.2.

According to Equation 3.2 and 3.4, the time complexity of frequency matrix computation is $O(H \cdot |d|^3)$. And the time complexity for sketch maintenance is $O(H \cdot |d|^2)$ for each incoming tweet, according to the analysis in Section 3.3.2.

It is not hard to maintain the sketch in parallel on multiple cores in one machine. Partition the stream D into S parts, i.e. $D = \cup_{s=1}^{S}\{D_s\}$. For each sub-stream D_s, we can maintain a sketch, which consists of two matrices $M_2^{(s)} = \mathcal{A}_t(\{f_{2,i}\}_{d_i \in D_s})$ and $M_3^{(s)} = \mathcal{A}_t(\{f_{3,i}\}_{d_i \in D_s})$. As $\mathcal{A}_t$ is in

[4]The Tweet NLP tool (http://www.ark.cs.cmu.edu/TweetNLP) is used.

fact a linear function, the sketch on stream D is equal to the sum of the sketch on each sub-stream D_s, i.e. $M_2 = \mathcal{A}_t(\{f_{2,i}\}) = \mathcal{A}_t(\cup_{s=1}^{S}\{f_{2,i}\}_{d_i \in D_s}) = \sum_{s=1}^{S} \mathcal{A}_t(\{f_{2,i}\}_{d_i \in D_s}) = \sum_{s=1}^{S} M_2^{(s)}$. The same, $M_3 = \sum_{s=1}^{S} M_3^{(s)}$. According to this, we maintain the sketch on multiple cores on a single machine. In particular, we build a process pool of size S, and each process in the pool is in charge of a sub-sketch. Notice that we do not partition the stream in advance. Instead, for each arriving tweet, randomly send it to one process in the pool on the fly.

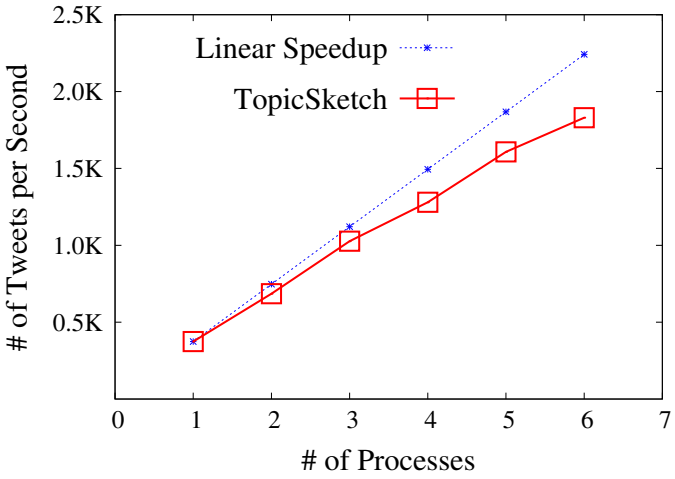


Figure 3.5: Comparison between throughputs for TopicSketch and Linear Speedup for varying numbers of processes.

To evaluate the throughput of the sketch maintenance and its scalability, we set the number of processes in the process pool from 1 to 6 respectively. Figure 3.5 shows the throughput for process pool of different sizes. Notice that when the number of the processes is 6, the throughput is 1,830 tweets per second (over one hundred and fifty millions tweets per day), which is on the same scale of the total number of tweets generated daily in the whole Twitter network. We can observe that the throughput linearly increases as the number of processes increases, which shows the scalability of TopicSketch. Also observe that the curve is a little away from the ideal case – linear speedup (the dashed line in Figure 3.5). It may be because of the additional cost for communication between processes.

3.4.1.2 Topic Inference

As described in Section 3.3.3, we can infer the bursty topics from H different sketches in parallel. In particular, we built a process pool and each process in the pool is in charge of performing *TensorDecompose* for one sketch under hash function $\mathcal{H}_h$. We set the size of process pool to 1 and 5, which represent the sequential version and fully parallel version respectively. We vary the number of topics K from 15 to 75. Figure 3.6 shows the performance of the algorithm. The result shows that although the sequential version gives an affordable running time (less than half minute), the parallel version provides significant improvement (10 seconds for 75 topics). Also observe that for both versions, as the number of topics increases, the inference time linearly increases, which supports our analysis about the time complexity of Algorithm 1, and shows the scalability of the inference algorithm.

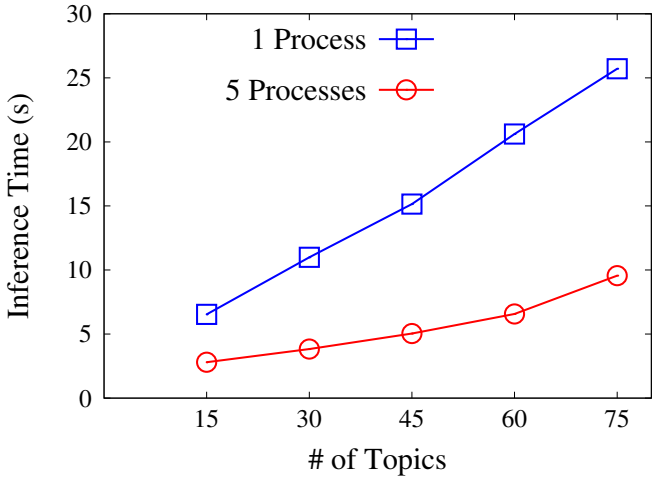


Figure 3.6: The inference time for varying numbers of topics.

3.4.2 Effectiveness Evaluation

3.4.2.1 Evaluation on Synthetic Data

We first use synthetic data to verify the correctness of our solution. As mentioned in Section 3.2.2, we model the tweet stream as a mixture of tweets of

different topics, and each tweet is only associated to one topic. We assign a temporal point process [42] and a distribution over words to each topic, then sample time stamps based on its point process, and sample tweets according to its distribution. We consider 3 types of topics: 1) the general topics (e.g. sport, education, car) which have steady arriving rates and take a large proportion in the whole stream; 2) the bursty topics which have highly intensive arriving rates after they go viral; 3) the noisy topics which are the small spikes in the stream. For general topics, the homogeneous Poisson point process is used to simulate the arriving rates of their tweets. And we draw the Poisson parameter λ from log-normal distribution to model different arriving rates for different topics. The Hawkes process [43] is used to simulate the arriving rates for bursty and noisy topics. Specifically, we adopt the following equation to calculate the arriving rate.

$$\lambda(t) = \lambda_0 e^{-(t-t_0)/w} + \alpha \sum_{t_i<t} e^{-(t-t_i)/w}$$

where t_0 is when the topic is generated, λ_0 is the initial intensity parameter and w is the decaying parameter. The second part in above equation mimics the self-enforcing phenomenon in Twitter, and exponential term models the waning of popularity over time.

In the synthetic data stream, we simulate 50 general topics, 1 bursty topic and 5 noisy topics. For each general topic, draw $\lambda \sim log\mathcal{N}(1,1)/60$ as its parameter. For bursty topic, the initial intensity λ_0 is set to 2.0. For the 5 noisy topics, λ_0 is set to 0.1, 0.2, ..., 0.5 respectively. For both bursty topic and noisy topics, $w = 100$ and $\alpha = 0.009$. All the topics are drawn from Dirichlet distribution. Particularly, we choose such noisy topics that the inner product of any noisy topic and the bursty topic is at least 0.01. And the vocabulary size is 5,000.

KL-divergence is used to compare the uncovered bursty topic with the

ground-truth bursty topic. The precision at top 25 words in the bursty topic is also calculated. We set different thresholds for detection. All experiments are repeated 500 times.

We compare the following methods.

- **Baseline**: the topic is estimated based on the acceleration vector of single word frequency, .i.e. $\mathcal{A}_t(\{f_{1,i}\})$.
- **SVD**: perform a SVD on the acceleration matrix of word pair frequency $\mathcal{A}_t(\{f_{2,i}\})$, and return the primary eigenvector.
- **TD**: perform the tensor decomposition in Algorithm 1, and return the topic with highest acceleration.
- **TD-CM**: tensor decomposition together with the dimension reduction technique Count-Min as shown in Algorithm 3.
- **TD-CM***: a variant of TD-CM, the subtle difference is that TD-CM* uses the second minimum value instead of the minimum value in Line 3 in Algorithm 2 for the purpose of robustness.

Notice that for the evaluation on synthetic data, no additional refining step is added. For TD-CM and TD-CM*, we use $H = 5$ hash functions, set the bucket size $B = 1000$ and pick the number of topics $K = 6$. Figures 3.7 (a) and (b) show the average performances for each of above methods. Figure 3.7 (c) shows the recall of our acceleration based detection method for varying thresholds. As expected, higher threshold leads to better performance, but lower recall. The same as many other threshold based solutions, we need to find a good trade-off between recall and precision.

To our surprise, we found that, after dimension reduction, TD-CM and TD-CM* even have better performances than the original TD. The reason is that the tensor decomposition method in Algorithm 1 relies on a "good" random vector η, such that for any $k_1 \neq k_2$, $\langle \eta, \phi_{k_1} \rangle \neq \langle \eta, \phi_{k_2} \rangle$ (See Appendix

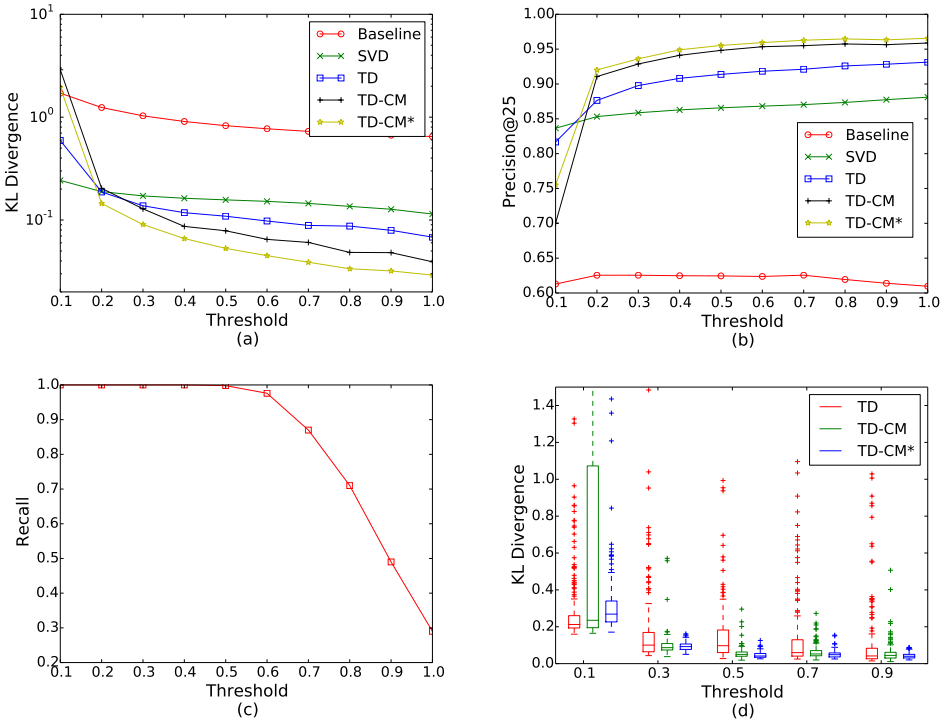


Figure 3.7: (a)-(b): Comparison of KL Divergence & Precision at top 25 words for varying thresholds; (c): Recall for varying thresholds; (d): KL Divergence for 200 different random vector η.

B). In high dimension space, there is higher chance to "violate" this condition than in lower dimension space. To verify this, on one synthetic data stream, we generate 200 random vectors, and see how different choices of η affect the performances. Figure 3.7 (d) illustrates that, although in average TD can get good performance, TD has much more outliers than TD-CM and TD-CM*. Considering the risk of failure due to a "bad" choice of η, in TD-CM* we always assume there maybe one copy of sketch in H fails to recover the bursty topic, and modify Count-Min by using the second minimum value instead of the minimum value in Line 3 in Algorithm 2. Figure 3.7 (d) shows TD-CM* has the most robust performance.

3.4.2.2 Evaluation on Real Data

In this section, we examine our solution on two real Twitter data sets: Singapore based data set and San Francisco based data set. Unlike synthetic data,

Twitter data set does not come with ground truth, i.e. the bursty topics. For evaluation, one indirect way is to identify bursty word pairs from tweet stream first (in a retrospective way), and then use these bursty word pairs as "ground truth" to measure the performance of our solution. The underlying logic is as follows. For each bursty word pair, we expect our solution can detect some bursty topic which contains this bursty word pair in its top words. On the other hand, for each detected bursty topic, we expect it contains some bursty word pair.

First, we count every word pair by day. Then for each word pair, we find the date when its daily count is far beyond its average. Particularly, we employ the following equation proposed in [82] to calculate the significant score for each word pair.

$$sig_\beta(c) = \frac{c - max\{\mu, \beta\}}{\sigma + \beta}$$

where c is the daily count of a word pair, μ is the average, σ is the standard deviation and β serves as a noise filter. $sig_\beta(c)$ here works like the z-score in the case of normal distribution. For a word pair (w_1, w_2) on some *date*, if its $sig_\beta(c) > 3$, then it is a bursty word pair on that date, and we generate a tuple $[date : (w_1, w_2)]$. After processing all the word pairs, we get a list of tuples. Based on this list, we measure the precision of our solution by the fraction of detected bursty topics that are "supported" by at least one tuple $[date : (w_1, w_2)]$, i.e. the bursty topic should contain (w_1, w_2) in its top 10 words, and it must be detected on *date*. For the recall, we get the top 1000 bursty word pairs according to their daily counts, and calculate the recall as the fraction of these bursty word pairs that are "reflected" in some detected bursty topic, which contains (w_1, w_2) in its top 10 words, and is detected on *date*.

In this experiment, we conduct the following baseline: once our detection system is triggered, train the LDA model using the recent 15 minutes tweets, and after Gibbs sampling, return the topic with largest number of assigned

words. We use 15 days tweets to tune parameters (e.g. threshold and number of topics) for both LDA and our solutions. Figure 3.8 shows the performances of different methods on both Singapore based tweets and San Francisco based tweets. TD-CM, TD-CM* are the same as Section 3.4.2.1. R-TD-CM* is the solution of TD-CM* followed by a refining step presented in Section 3.3.3. It shows that, overall our solutions outperform LDA. Further more, TD-CM, TD-CM* and R-TD-CM* have roughly the same recall, and because of the refining step, R-TD-CM* gets higher precision than TD-CM and TD-CM*.

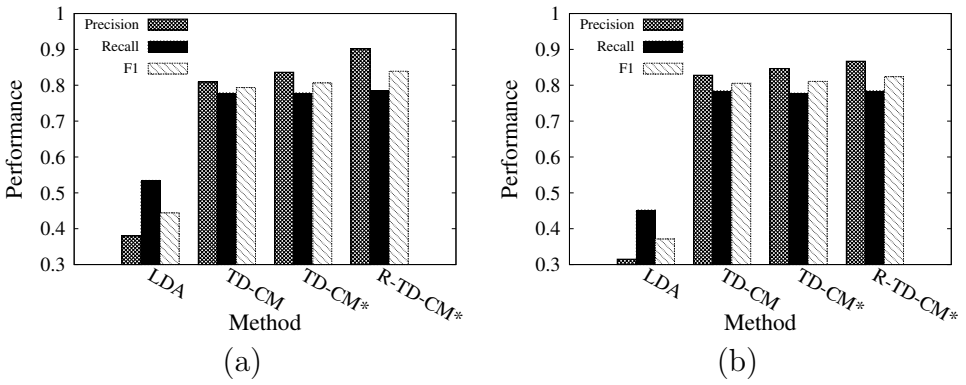


Figure 3.8: Performance on (a) San Francisco data and (b) Singapore data.

The above precision which is measured by bursty word pairs actually cannot fully reflect the quality of detected topics. A topic which contains some bursty word pair may also have other irrelevant words as its top words. To measure the coherence of the detected bursty topics, we adopt the *word intrusion* task proposed in [14]. In this task, the subject is presented with six randomly ordered words, in which five words are from a detected bursty topic and the other one is a *intruding* word. The task of the user is to identify the *intruding* word which is not related to this bursty topic. Different from the task in [14], if only presented with the words, it is not easy for human user to find out the intruding word. For instance, when presented with {"spurs", "horse", "thunder", "99-107", "game", "fisher"}, without exploring the related tweets, a human user, who is not familiar with USA basketball, may not know this set of words (except the intruder "horse") actually is about a basketball match,

consequently, she may identify a wrong intruder. So in our task, the words are presented together with an interactive plot which shows the counts of each word over time, and the tweets contains these words, within a time window around the detection time. (See a demo here[5].) After exploring all the above information, if the presented set of words lacks coherence, the human user would be still confused, and just pick an intruder at random.

For each detected bursty topic, we first select five words with highest probabilities from it. Then the intruding word is selected at random from a pool of words which are popular in the tweet stream on the detection date. We chose 50 bursty topics randomly for each method. Three human users are asked to perform this task. We calculate the coherence score as $\sum_{i=1}^{50}\sum_{j=1}^{3}\mathbf{1}(intruder_i = word_{ij})/150$, where $intruder_i$ is the intruding word for the i-th bursty topic, $word_{ij}$ is the chosen word from human user j for the i-th bursty topic. Figure 3.9 shows the comparison between different methods in terms of coherence score.

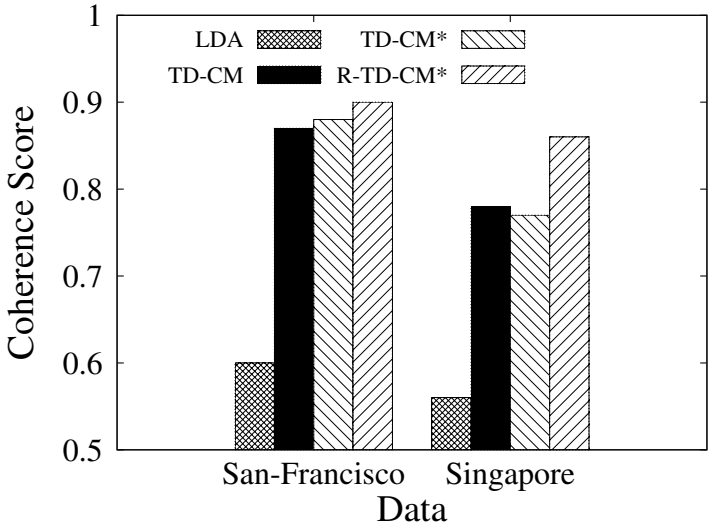
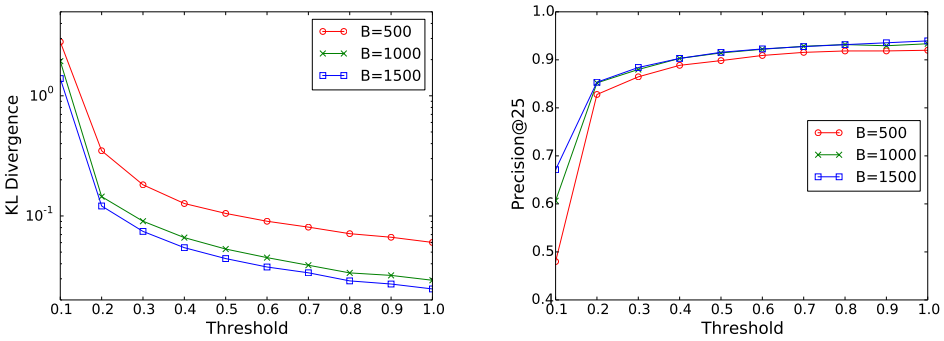


Figure 3.9: The coherence scores of the detected bursty topics from different methods.

[5] http://topicsketch.appspot.com/

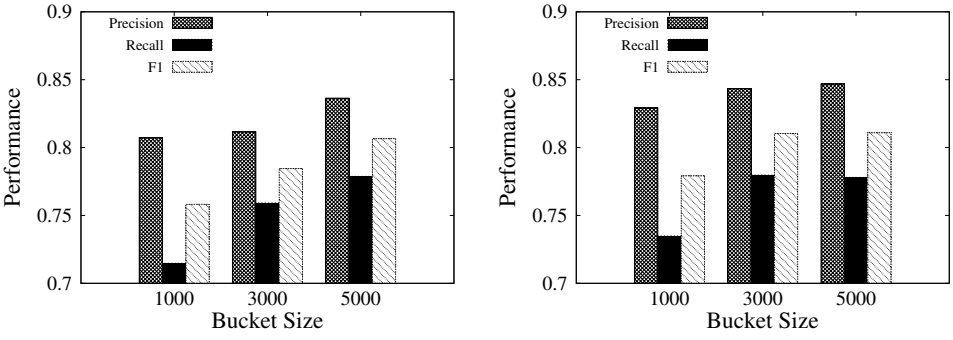
3.4.2.3 Parameter Analysis

Bucket Size. Here we study the effect of bucket size B. Figure 3.10 shows the performances of TD-CM* for different bucket sizes on synthetic data. We can observe the improved performance as bucket size B increases. From $B = 500$ to $B = 1000$, the improvement is significant, while from $B = 1000$ to $B = 1500$, the improvement is little. Figure 3.11 shows how the bucket size B affects the performance of TD-CM* on real data. As expected, it shows that better performance can be achieved when we set bigger bucket size B. However, bigger bucket size means higher computational cost. Figure 3.6 shows the computational cost is acceptable when $B = 5000$. So in this work, rather than explore other even bigger bucket size, we empirically set $B = 5000$.



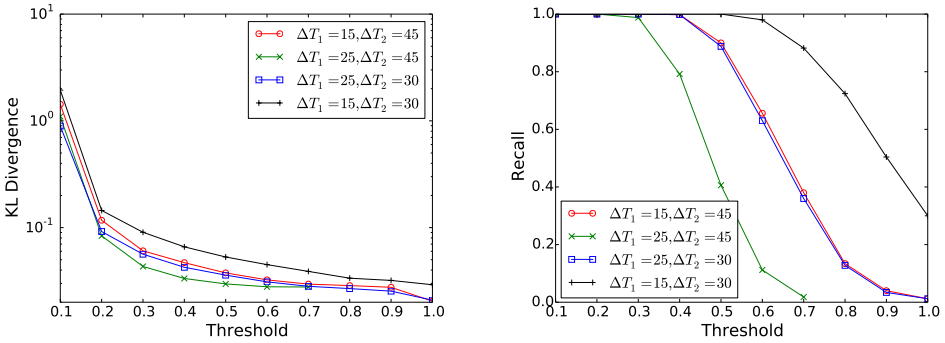
(a) (b)

Figure 3.10: Varying bucket size B, performances on synthetic data: (a) KL Divergence and (b) Precision at top 25.



(a) (b)

Figure 3.11: Varying bucket size B, performances on (a) San Francisco data and (b) Singapore data.



(a) (b)

Figure 3.12: Varying smoothing parameters ΔT_1 and ΔT_2, performances on synthetic data: (a) KL Divergence, (b) Recall.

Time Window. We also check the effect of smoothing parameters ΔT_1 and ΔT_2 on synthetic data. Figure 3.12 (a) (b) show how different settings of ΔT_1 and ΔT_2 affect the performance of TD-CM* in terms of KL divergence and recall. Basically, bigger smoothing parameters leads to higher precision but lower recall. In practice, we test different settings of smoothing parameters, and choose the one which provides a good balance between precision and recall.

Topic Number. As shown in Section 3.2.1, TopicSketch is designed to use acceleration to filter out the large number of general topics such as food, car and education. So in the sketch, only bursty topics are preserved (See Figure 3.2). This is why small number of topics is enough for inferring the bursty topics from sketch. In our experiment, we use synthetic data to verify this. In the synthetic data, there are 50 general topics, 1 bursty topic and 5 noisy topics. The following Figure 3.13 shows that the best choice of K is actually 5 or 6, which is much smaller than the total number of general topics, i,e. 50. For the evaluation on real data, we also found that small number of K is the enough for discovering bursty topics.

Parameter Tuning. In order to find the best set of parameters, it usually requires burdensome and systematic experiments. And for picking the best set of parameters, an objective should be defined. However, from a system perspective, we face several contradictory objectives — the inference time,

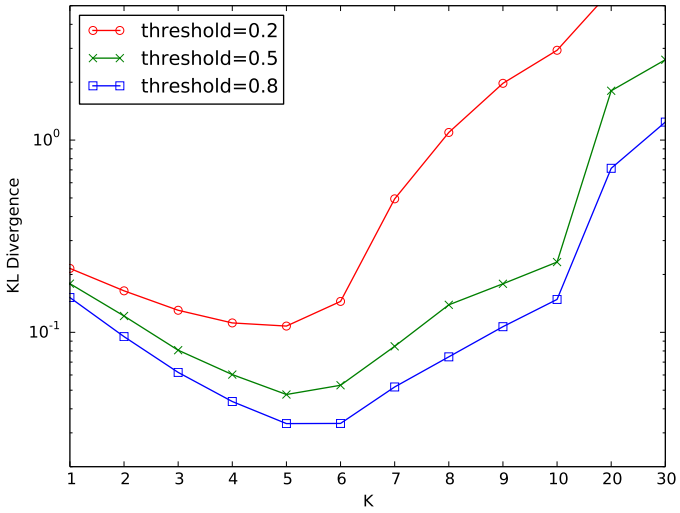


Figure 3.13: Varying topic number K, $B = 1000$, on synthetic data.

computational space cost, precision, recall, and detection delay. For example, as shown in Figures 3.10 and 3.11, as we increase the bucket size B, we get higher precision and recall. However, at the same time, we have longer inference time because the sketch becomes larger. A practical way is to optimize a single objective and maintain other objectives within certain ranges. In this work, we set F_1 score (which considers both precision and recall) as the main optimization objective, and at the same time, make sure other objectives within reasonable ranges. We use a hold-out dataset for parameter tuning. Because of the large search space and the huge size of data, simply using grid search is very time-consuming. So we heuristically set the parameters as follows. First, we test different bucket sizes and pick the biggest one with which the computational cost we can afford. Then, we test different thresholds, topic numbers and window sizes in a low-resolution grid and pick the best set of parameters. Finally, fixing the topic number and window sizes, we search the best threshold in a high-resolution grid.

Refining Process. In order to prevent hash collision as well as the injection from spam accounts, we propose a heuristic step to refine the inferred topics. This step is optional. However, a proper refining step can improve the overall

quality of the detected topics (See Figures 3.8 and 3.9). In our experiment, we empirically set the parameters in this step as follows: (1) the threshold of trivial topic filtering: 20; (2) the threshold of noisy topic filtering: 6; (3) the threshold of spam filtering: 25; (4) the time window of rare word pruning : 30 minutes. Because too many parameters here, we empirically test few different settings of these parameters, and choose the one which provides a good balance between precision and recall.

3.4.3 Comparison with other solutions

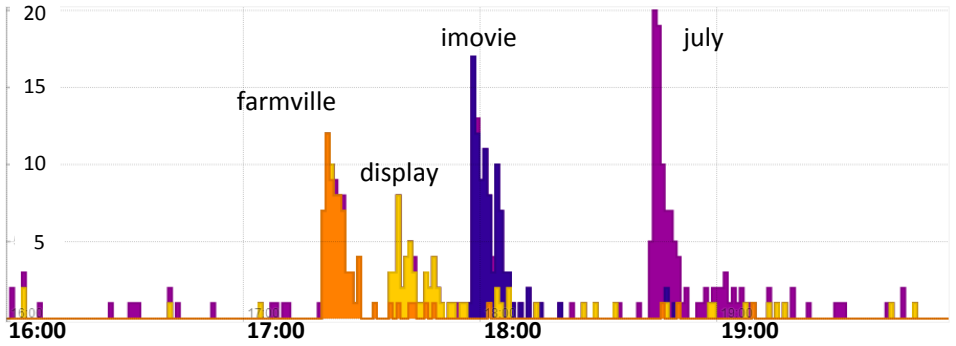
3.4.3.1 Comparison with Twevent

Table 3.1: List of events detected by TopicSketch and Twevent

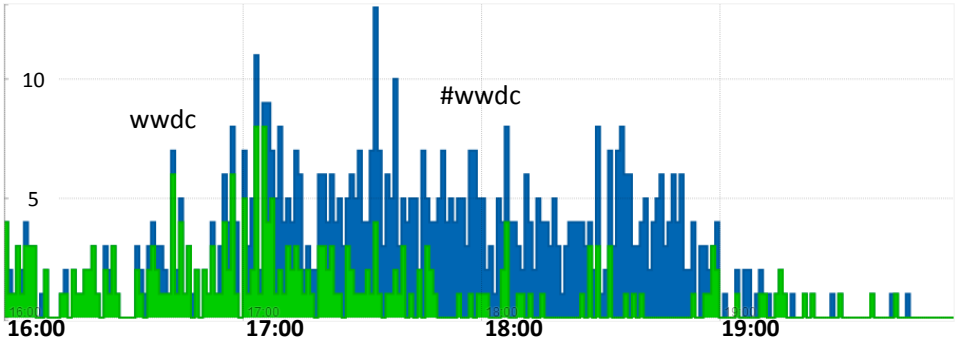
Event	Sub-Event	TopicSketch	Twevent
WWDC	Farmville client for iPhone 4 was demonstrated.	#wwdc, farmville, iphone, zynga, netflix, god, ipad, wwdc, comes, soon	steve jobs, imovie, wwdc, iphone, wifi
	Retina display of iPhone 4 was introduced.	iphone, #wwdc, retina, display, pixels, crystal, clear, 326, space, interesting	
	iMovie for iPhone 4 was demonstrated.	iphone, #wwdc, imovie, 720p, sensor, 3gs, apple, 30fps, gonna, illuminated	
	New iPhone 4 was available in Singapore in July.	iphone, singapore, july, launching, coming, 4g, available, release, #wwdc, early	

We compare TopicSketch to previous Twitter event detection system Twevent [65] by case studies. We adopt the same dataset as used in the original paper [93]. We present in Table 3.1 an event detected by both algorithms[6] – Apple WWDC 2010, to show the differences between TopicSketch and Tewent. We manually group together sub-events belonging to a single larger event. Table 3.1 demonstrates TopicSketch's ability to describe events at a finer granularity. On June 7, 2010, Steve Jobs announced the release of the fourth generation iPhone, causing huge wave in Twitter. At this WWDC, several new features of this new generation iPhone were introduced, including farmville client, retina display and iMovie. As shown in Table 3.1, TopicSketch not only detected the big event of Apple WWDC 2010 as a whole, it also detected a sequence of

[6]We referred the results listed in Table 2 from [65].



(a)



(b)

Figure 3.14: Apple WWDC 2010: (a) keywords for each sub-event (b) keywords for the general event WWDC.

sub-events. Along timeline, we show in Figure 3.14(a) the number of tweets per minute which contain a keyword of each sub-event, in Figure 3.14(b) the number of tweets per minute which contain the keywords of this event – wwdc and #wwdc. From this figure we can see the duration of the burst of "wwdc" is quite long for one day period, which is a big trend. So it is significant enough to be detected by both TopicSketch and Twevent. Compared with "wwdc", the duration of the burst of each sub-event is relatively short, which is about 15 minutes. The keywords such as "display" appeared and disappeared in a short time period and it is not significant enough considered against a longer time window to be detected by Twevent. As the sketch captures the acceleration of the tweet stream which reflects real-time data dynamics, TopicSketch successfully detected them. In particular, each peak in the figure which triggers our system indeed corresponds to a highlight of the WWDC event, which are some of the features as they were being announced, and from which we can

tell those new features of iPhone that users find more interesting than others.

3.4.3.2 Comparison with SigniTrend

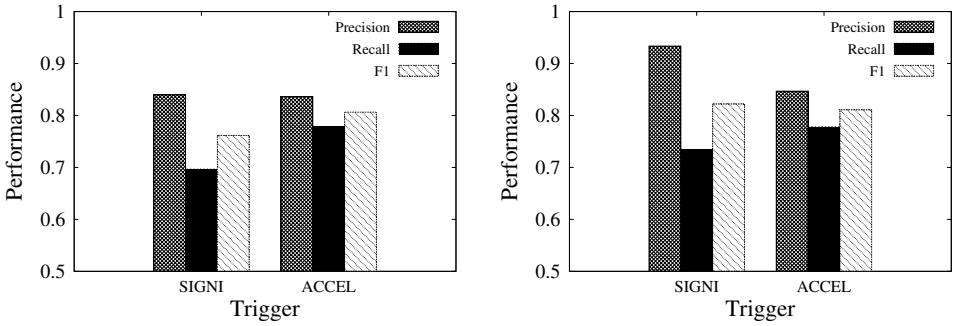


Figure 3.15: For different triggers, performances of TD-CM* on (a) San Francisco data and (b) Singapore data.

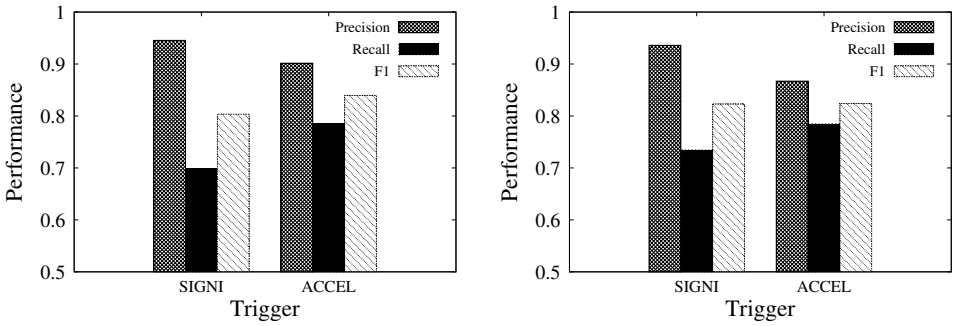


Figure 3.16: For different triggers, performances of R-TD-CM* on (a) San Francisco data and (b) Singapore data.

SigniTrend is the state-of-the-art emerging topics detection solution [82]. SigniTrend performs a two-stage detection: first detects the bursty words (and word-pairs) whose significance score exceed a predefined threshold on the fly, and then performs end of day analysis to cluster the bursty words into larger topics. Similarly, TopicSketch also first detects the bursty word-pairs according to their accelerations on the fly, and once the system is triggered immediately infer the bursty topics from the sketch. As the detail of clustering step is not presented in [82], we do not directly compare the results from these two solutions. Instead, we check how different triggers affect the performance of our

solution. Particularly, we use the significance score from SigniTrend instead of the acceleration proposed in our dissertation as the trigger, and the inference part after triggering remains the same. We also use 15 days tweets to tune the parameters. Figure 3.15 and Figure 3.16 show the performances of TD-CM* and R-TD-CM* for different triggers respectively. (SIGNI stands for significance score and ACCEL for acceleration). It can be observed that, roughly, using significance score as the trigger can achieve higher precision while using acceleration as the trigger can achieve higher recall.

3.4.3.3 Comparison with Twitter trending topic

Table 3.2: List of events detected by TopicSketch and Twitter on April 8, 2013

Event	**First detected by TopicSketch**	**First appeared in Twitter**
The popular program WrestleMania was discussed.	None	#wrestlemania **(14:34:12)**
Fans asked Luke Brooks (@luke_brooks) to follow them.	@luke_brooks, #followmeluke, follow (19:06:51)	#followmeluke **(18:54:12)**
The sudden demise of Margaret Thatcher.	thatcher, margaret **(20:05:51)**	margaret thatcher (20:49:12)

The trending topics provided by Twitter API are usually represented by single words, hashtags and phrases. Our list of trending topics is obtained at a frequency of every 5 minutes from the beginning of 2013. To demonstrate the difference between the trending topics as presented by Twitter and the bursty topics we are interested in, which are two related yet different concepts, we focus on the day of April 8, 2013, when the former British Prime Minister Margaret Thatcher died. We enumerated all the trending topics of Twitter for this day. By verifying bursty events and filtering out those that are not (including those all-time popular topics), we found three events as shown in Table 3.2. In Figure 3.17, we show the number of tweets per minute which contain the keyword of each event and the time stamp when TopicSketch and

Twitter each detected them. Our observation is that: the more bursty the event is, the better TopicSketch performs. As there was no obvious burst, TopicSketch missed the event of *wrestlemania*. For the event of *followmeluke*, Twitter is about 10 minutes faster than TopicSketch. However, for the big breaking event of *thatcher*'s demise, TopicSketch is about 40 minutes faster than Twitter, and only about 15 minutes later than the first relevant tweet in our dataset.

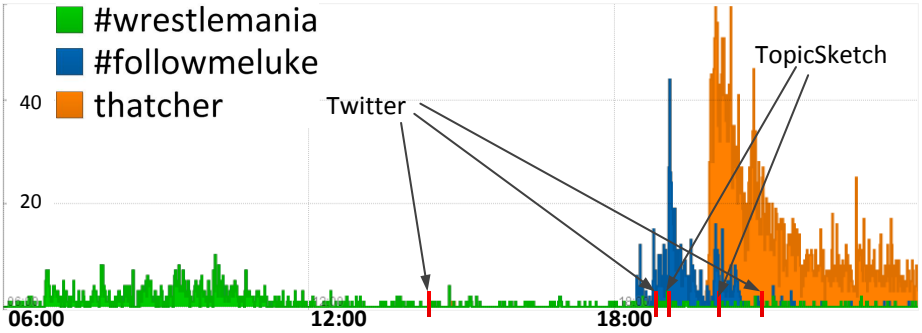


Figure 3.17: Topics detection from TopicSketch and Twitter.

3.4.3.4 Limitation of TopicSketch

Compared with Twevent and SigniTrend, TopicSketch is fragile when facing spam accounts. For instance, some spam account may abuse key word "music" by injecting a lot of tweets about some music album into tweet stream in a very short period of time. This would produce words with significant acceleration, and therefore trigger our detection system. After the topic inference step, our system would report a topic about this music album, which is not popular at all. Figure 3.18 illustrates the volume of tweets which contain a spam word in one day. A burst roughly every 4 hours can be identified. Thus in Section 3.3.3, a refining step is proposed to remedy this situation.

In the experiment, we also found that TopicSketch may miss continual topics, which represent events with long durations. For example, Figure 3.19 shows the number of tweets per minute which contain the keywords of two topics – one bursty topic (the performance of Infinite) represented by keyword

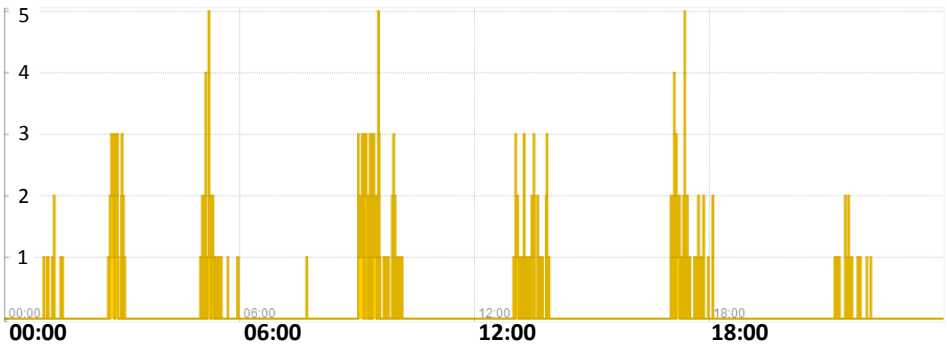


Figure 3.18: Detected bursty topic created by spam.

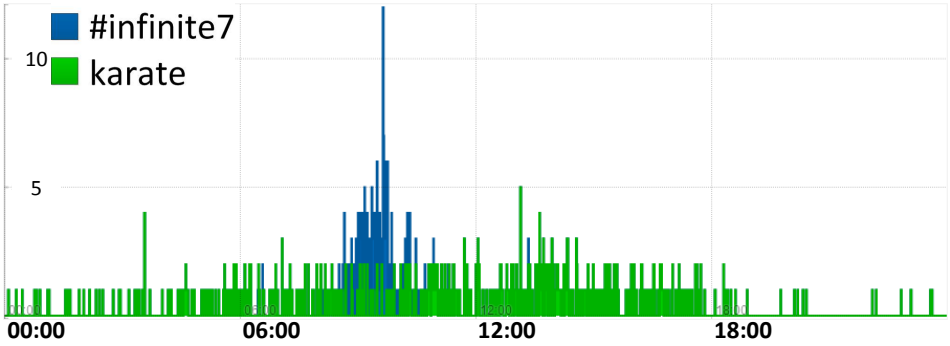
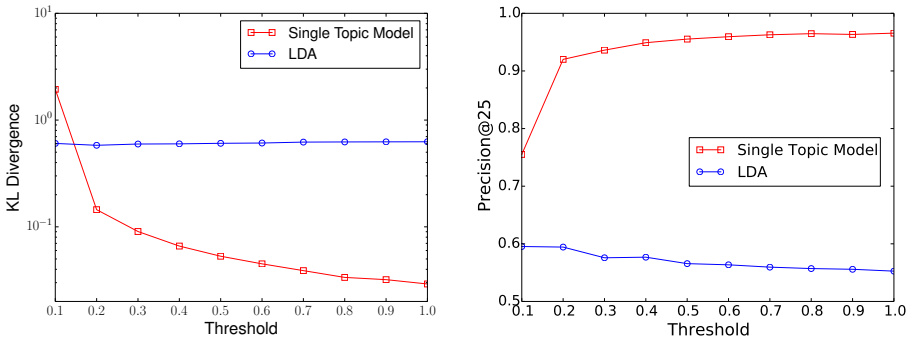


Figure 3.19: Bursty topic vs continuous topic.

#infinite7 and one continual topic (the release of "The Karate Kid") represented by keyword *karate*. TopicSketch detected the former while Twevent missed it. In the meantime, Twevent detected the latter, while TopicSketch missed it. Form this figure, we can see that for *karate*, there was no obvious burst on the timeline as it was being continually discussed. In contrast, for *#infinite7*, we can see a major burst over a few hours along the timeline. It is clear that, due to its daily-base detection, Twevent is good at detecting events that are continually discussed over a long period of time, but may miss events with shorter bursts. On the other hand, one would arrive almost the opposite conclusion for TopicSketch.

Besides, we also found that, due to the single topic assumption in Section 3.2.2, TopicSketch is not suitable for stream of long texts with multiple topics. First, longer text would take more time. As shown in Section 3.4.1.1, the complexity of frequency matrix computation is $O(H \cdot |d|^3)$. For instance, if the processing time for a 10-words tweet is t, a text with length 100 would take

time $1000 \cdot t$, which is expensive. Second, to check the effectiveness, we run our algorithm TD-CM* on a synthetic data which is generated from LDA model. Figure 3.20 presents the performance of TD-CM* on texts from Single Topic Model and LDA. It shows that our method does not work well on long texts from LDA model.



(a) (b)

Figure 3.20: Performance on Single Topic Model texts and LDA texts.

3.5 Summary

In this chapter, we propose TopicSketch a framework for real-time detection of bursty topics from Twitter. Due to the huge volume of tweet stream, existing topic models can hardly scale to data of such sizes for real-time topic modeling tasks. We develop a "sketch of topic", which provides a "snapshot" of the current tweet stream and can be updated efficiently. Once burst detection is triggered, bursty topics can be inferred from the sketch efficiently. Compared with existing event detection system, from a different perspective – the "accelerations of topics", our solution can detect bursty topics in real-time, and present them in finer-granularity.

Chapter 4

Modelling Cascades Over Time in Microblogs

In this chapter, we predict how cascades grow from a micro perspective – model how the chance of each particular user joins a cascade changes over time. By simulating each user's behavior, we forecast the future cascade size. Particular, we first build a general time-aware cascade model for each particular cascade, in which the chance of one user's re-sharing behaviour over time is modelled as a hazard function of time. Then based on two key observations on user retweeting behaviour, we design an appropriate hazard function specifically for Twitter network. Finally, we evaluate our simulation based approach on a real Twitter dataset with over two million retweeting cascades.

4.1 Time-aware Cascade Model

In this section, we describe a general time-aware cascade model from the aspect of one particular cascade, which is based on previous time-aware cascade models such as [37] [70] [39]. We consider a general social network $G =< U, E >$, where U represents the set of users, and E is a set of directed links between users of U (See Figure 4.1), each representing the channel through which information in a cascade could flow in the directions as indicated. For example

in Twitter, if u_i follows u_j, there is a directed link going from u_j to u_i, denoted as (u_j, u_i). To study cascades, given any user u_i, we are interested in the set of users whose information can potentially reach u_i in cascades, which we call u_i's *followee set* and denote as $Followee^{(i)} = \{u_j | (u_j, u_i) \in E\}$. Similarly, we also care about the set of users who can potentially receive information from u_i, which we call u_i's *follower set*, and denote as $Follower^{(i)} = \{u_j | (u_i, u_j) \in E\}$. For a cascade of any message, we denote as u_0 the original source and use a random variable T_i to denote the timestamp when user u_i re-shares the message. Correspondingly, t_i is the observation of random variable T_i. As a trivial case, $T_0 = t_0$ with probability 1 and $t_i = \infty$ if u_i never re-shares the message.

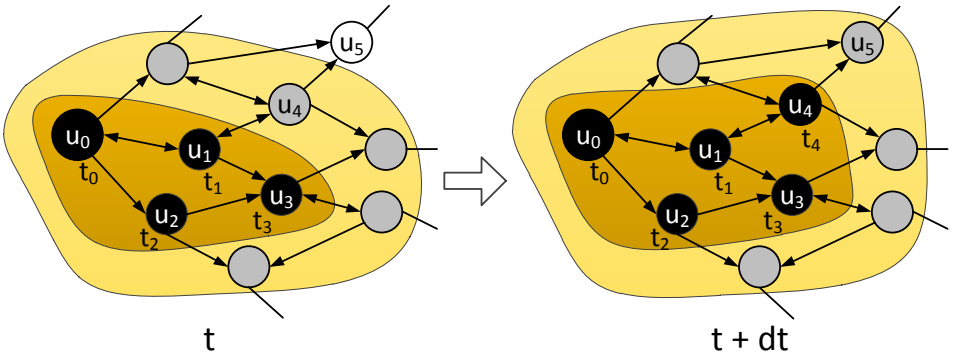


Figure 4.1: A snapshot in the development of a cascade (from timestamp t to $t + dt$).

As illustrated in Figure 4.1, at any timestamp in the cascade development, we identify three types of nodes: (I) **Black Nodes**: the users who have already re-shared the information, with their re-share timestamps; (II) **Grey Nodes**: the users who are exposed to the information, yet have not re-shared it; and (III) **Blank Nodes**: the users who yet to be exposed to the information. As the cascade develops, Blank Nodes could become Grey Nodes (e.g., u_5), which in turn could become Black Nodes (e.g., u_4).

Intuitively, we model the development of a cascade as a stochastic process as follows. Suppose at timestamp t_0, the information source u_0 posts a piece of information. At each following timestamp t, we take a probabilistic point of view similar to Cascade Model toward the growth of the cascade. The key

is to focus on the Grey Nodes as they are the only ones to potentially re-share the information in the next small time slice $(t, t+dt]$. We denote the Grey Nodes as $\mathbb{X}(t)$. At time $t+dt$, $\mathbb{X}(t)$ can be divided into two parts: (I) those who have shared the information by $t+dt$ (i.e., they become Black Nodes), which is denoted as $\mathbb{X}^{(1)}(t)$, and (II) the rest, which is denoted as $\mathbb{X}^{(2)}(t)$, i.e., they remain Grey Nodes, $\mathbb{X}^{(2)}(t) = \mathbb{X}(t) \setminus \mathbb{X}^{(1)}(t)$.

At any timestamp t, a cascade can be represented by a set of pairs $\mathbb{C}(t) = \{< u_i, t_i > | t_i \leq t\}$, e.g., $\mathbb{C}(t) = \{< u_0, t_0 >, < u_1, t_1 >, < u_2, t_2 >, < u_3, t_3 >\}$ in Figure 4.1. Similarly, $\mathbb{C}(t, t+dt) = \{< u_i, t_i > | t < t_i \leq t+dt\}$, e.g., $\mathbb{C}(t, t+dt) = \{< u_4, t_4 >\}$ in Figure 4.1. Denote $P_i(t)$ as the probability that user u_i re-shares the information in time slice $(t, t+dt]$ given she has not re-shared the information by time t. (It is worth noting that $P_i(t)$ is information specific, see Equation 4.2 below.) Then the probability that the cascade grows from $\mathbb{C}(t_0)$ into $\mathbb{C}(t)$ can be defined as the following recursive equation.

$$\begin{cases} P(\mathbb{C}(t+dt)) = P(\mathbb{C}(t+dt)|\mathbb{C}(t)) \cdot P(\mathbb{C}(t)) \\ P(\mathbb{C}(t_0)) = 1 \\ P(\mathbb{C}(t+dt)|\mathbb{C}(t)) = \prod\limits_{u_i \in \mathbb{X}^{(1)}(t)} P_i(t) \cdot \prod\limits_{u_{i'} \in \mathbb{X}^{(2)}(t)} (1 - P_{i'}(t)) \end{cases} \tag{4.1}$$

Equation 4.1 defines the entire stochastic process of a growing cascade, which only depends on the probability $P_i(t)$. From previous analysis, it is not hard to see that $P_i(t)$ depends on the users in her followee set who have already shared this piece of information by time t, which is denoted as $Followee^{(i)}(t) = \{u_j | t_j \leq t, u_j \in Followee^{(i)}\}$. It follows that $P_i(t) = P(t < T_i \leq t+dt | T_i > t, \{T_j = t_j\}_{u_j \in Followee^{(i)}(t)})$. We model this probability using the following hazard function[1].

$$P_i(t) = h_i(t, \{t_j\}_{u_j \in Followee^{(i)}(t)}; \mathbf{\Theta}) \cdot dt \tag{4.2}$$

[1]See Appendix D for the formal definition of hazard function.

where $\mathbf{\Theta}$ are the parameters which are related to the original information posted by u_0 and, reflect how tensely this piece of information gets the interest of public and how quickly users react to it. If at time t for user u_i, $Followee^{(i)}(t)$ is an empty set (e.g. the blank node u_5 at time t in Figure 4.1), then just let $h_i(t, \{t_j\}_{u_j \in Followee^{(i)}(t)}; \mathbf{\Theta}) = 0$.

To apply this general time-aware cascade model to any particular social network, one only has to identify the appropriate hazard function in Equation 4.2 and the parameters $\mathbf{\Theta}$, after which the stochastic process would be fully defined. So the crucial thing here is to design a proper formula for the hazard function in Equation 4.2, which really fits the real cascades.

4.2 Model Application: Twitter

In this section, we show how the general time-aware model in Section 4.1 can be applied to the concrete setting of Twitter to model cascades of tweets. As mentioned in Section 4.1, the key issue is to design the appropriate hazard function in Equation 4.2. For this, we first observe how a tweet is retweeted in Twitter, then give criteria for the hazard function, at last propose the the appropriate hazard function in Twitter.

4.2.1 Observations

In Twitter, we identify the following two observations which affect how a tweet would be retweeted in a cascade, and accordingly provide us the clues to design the appropriate hazard function in Equation 4.2.

Observation 1. *Only the first re-sharer matters.* For any user, only the tweet from the re-sharer who first re-shares it would appear in the user's home timeline. For example, in Figure 4.1, suppose u_1 re-shares the tweet from u_0 earlier than u_2, i.e., $t_1 < t_2$. Then although both u_1 and u_2 are in the followee set of u_3, only the tweet re-shared by u_1 appears in u_3's home timeline

at time t_1. Based on this observation, we conclude that only the first re-sharer in a user's followee set would affect her subsequent retweeting behaviour. Consequently, we have the following simpler formula (Equation 4.3) instead of the general formula in Equation 4.2.

$$P_i(t) = h_{i,j^\star}(t, t_{j^\star}; \mathbf{\Theta}) \cdot dt \tag{4.3}$$

where $j^\star = argmin_j\{t_j | u_j \in Followee^{(i)}(t)\}$.

Observation 2. *The chance of a tweet to be retweeted decreases as time goes by.* As more recent tweets appear higher in a user's home timeline and is more likely to attract user's attention, the chance of a tweet to be retweeted decreases as it sinks down along the timeline. Based on this observation, we further refine the formula as follows in Equation 4.4.

$$P_i(t) = h_{i,j^\star}(t - t_{j^\star}; \mathbf{\Theta}) \cdot dt \tag{4.4}$$

where $h_{i,j^\star}(\tau, \mathbf{\Theta})$ is a decreasing function of τ ($\tau = t - t_{j^\star} > 0$). It is worth noting that not all the hazard functions are decreasing. Take the risk of death as an example, as people become older and older, the risk of death in fact becomes higher and higher.

Besides, as a trivial case, when $t \leq t_{j^\star}$, u_i is not exposed to the tweet from the original user u_0, so there is no chance for u_i to retweet this tweet. And when $t > t_{j^\star}$, u_i have a chance to read the tweet from her own home timeline, and it is always possible for u_i to retweet it. Naturally we have the following constraints in Equation 4.5.

$$\begin{aligned} h_{i,j^\star}(\tau; \mathbf{\Theta}) &= 0, (\tau \leq 0) \\ h_{i,j^\star}(\tau; \mathbf{\Theta}) &> 0, (\tau > 0) \end{aligned} \tag{4.5}$$

4.2.2 Hazard Function Design

The analysis in Section 4.2.1 leads to the intuition that the appropriate hazard function in Twitter setting should be a decreasing function $h_{i,j^\star}(\tau;\mathbf{\Theta})$. In this subsection, we develop the concrete formula of $h_{i,j^\star}(\tau;\mathbf{\Theta})$. First, we make a simplification to replace $h_{i,j^\star}(\tau;\mathbf{\Theta})$ by $h(\tau;\mathbf{\Theta})$, which means no matter who u_i is and who is the first re-sharer $u_{j^\star}$, their hazard functions share the same formula $h(\tau;\mathbf{\Theta})$. To keep our notations simple without any loss in meaning, we just omit the parameter set $\mathbf{\Theta}$, and the hazard function is simply denoted as $h(\tau)$.

As $H(\tau)$ is the integration of $h(\tau)$, according to Equation 4.5, we have $H(0) = 0$ and $H(\tau)$ should be a increasing function of τ. Besides, another fact is that if user u_i hasn't retweeted the tweet from u_0, u_i's home timeline will be full of other new incoming tweets, and u_i will never re-share it. It means $F(\infty) < 1$, and consequently $H(\infty) = -log(1-F(\infty)) < \infty$. Based on all the analysis in Sections 4.2.1 and 4.2.2, we list the constraints for $H(\tau)$ in Table 4.1.

1. $H(0) = 0$.
2. $H(\infty) = -log(1 - F(\infty)) < \infty$.
3. $H(\tau)$ is an increasing function of τ.
4. $h(\tau) = \frac{\mathrm{d}H(\tau)}{\mathrm{d}\tau}$ is a decreasing function of τ.

Table 4.1: Hazard Function Constraints for Twitter.

Denote $\lambda = H(\infty)$ as one of the parameters. According to these constraints, our observations from the data (see Figure 4.3 (b) below) and the insight of human behaviour from other work [7], we eventually propose the following heavy-tail hazard functions $H(\tau)$ and $h(\tau)$ for Twitter setting.

Hazard Function For Twitter Cascades:

$$H(\tau) = \lambda \cdot (1 - (\frac{\tau}{\alpha} + 1)^{-\beta}) \tag{4.6}$$

$$h(\tau) = \frac{\mathrm{d}H(\tau)}{\mathrm{d}\tau} = \lambda \cdot \frac{\beta}{\alpha} \cdot (\frac{\tau}{\alpha} + 1)^{-(\beta+1)} \tag{4.7}$$

where $\lambda > 0$, $\alpha > 0$, $\beta > 0$. Here the parameter set $\mathbf{\Theta} = \{\lambda, \alpha, \beta\}$. In fact, our experiment shows that the probability that one user who is exposed to one tweet actually retweets it at last is quite small (around 0.01), which means $F(\infty)$ is near to 0 and $F(\infty) \approx H(\infty) = \lambda$. So the parameter λ describes the eventual re-tweeting probability. The larger λ is, the larger is the proportion of users who re-tweet one particular tweet in the users who are exposed to it. According to Equation 4.7, the parameter α describes the scale of hazard function $h(\tau)$, and the parameter β describes the shape of $h(\tau)$, which are similar to the scale parameter and shape parameter of Weibull distribution.

4.2.3 Hazard Rate Illustration

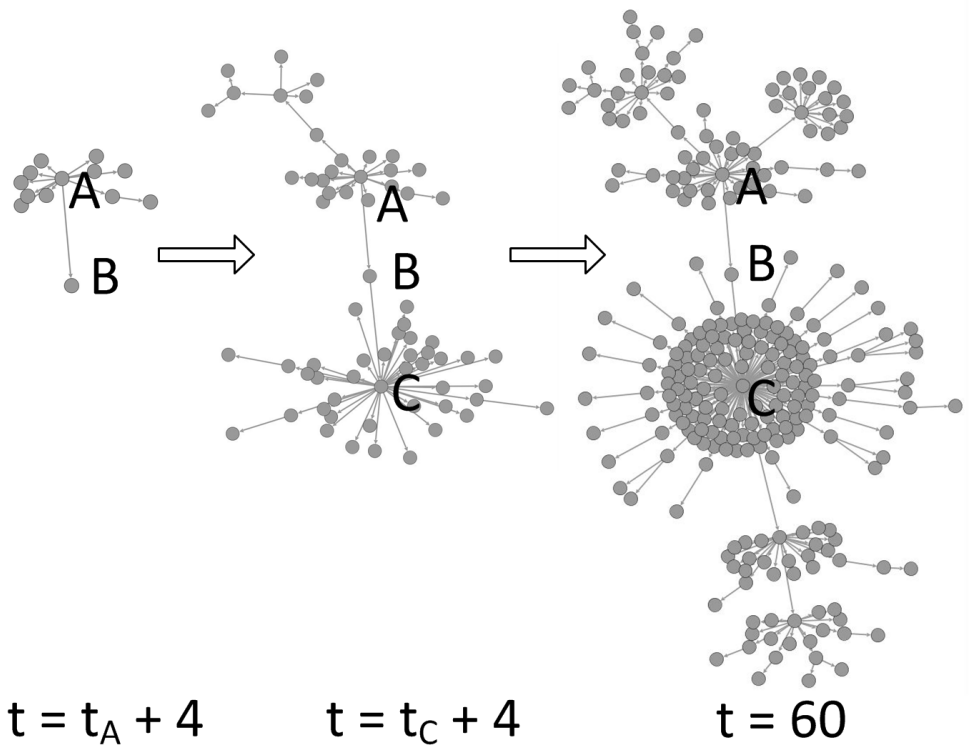


Figure 4.2: A Real Cascade Example.

We use a real cascade example here to illustrate how the proposed hazard function fits the hazard rate in real data. Section 4.4 presents more comprehensive evaluation including how different choices of hazard function affect the fitting performance of the model.

Figure 4.2 shows the growing process of a real cascade of a local news tweet — The tweet was initiated from node A the original source, and passed through node B before reaching node C which has the largest degree and has played the most important role in triggering the dramatic growth of the cascade. This is also demonstrated by Figure 4.3 (a) which shows the retweeting rate of this cascade, i.e., the number of users who have retweeted per minute. The spike corresponding to the greatest drive to the retweeting rate happened at timestamp t_C, after user C retweeted the message.

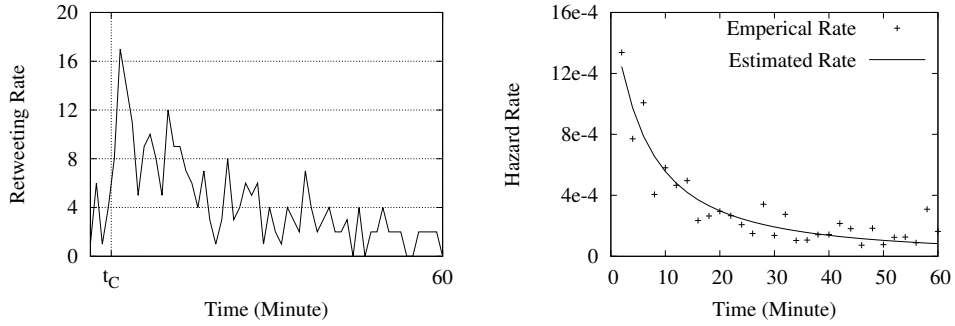


Figure 4.3: (a) Retweeting Rate Per Minute. (b) Hazard Rate Comparison.

We calculate the empirical hazard rate of this cascade as follows. Two sets of users are involved: (I) the set of users who have retweeted the tweet by time T, i.e., $\mathbb{C}(T)$ and (II) the set of users who have been exposed to the tweet but haven't retweeted it yet, i.e., $\mathbb{X}(T)$, where T is long enough for estimating the empirical hazard rate. For user u_i in $\mathbb{C}(T)$, we calculate $\tau_i = t_i - t_{j^\star}$, where $j^\star = argmin_j\{t_j | u_j \in Followee^{(i)}(t)\}$. τ_i measures how long it takes for u_i to retweet the tweet after being exposed to it. For user u_i in $\mathbb{X}(T)$, we calculate $\tau_i = T - t_{j^\star}$, which measures how long u_i has been exposed to the tweet. According to the definition of hazard rate (see Appendix D), the empirical

hazard rate of this cascade is calculated by using the following Equation 4.8.

$$\begin{aligned}\hat{h}(\tau) &= \frac{\hat{P}(\tau < T \leq \tau + dt | T > \tau)}{dt} \\ &= \frac{|\{u_i | u_i \in \mathbb{C}(T), \tau < \tau_i \leq \tau + dt\}| + 1}{|\{u_i | u_i \in \mathbb{C}(T) \cup \mathbb{X}(T), \tau < \tau_i\}| + 2} \cdot \frac{1}{dt}\end{aligned} \tag{4.8}$$

Figure 4.3 (b) shows the empirical hazard rate and the estimated hazard rate based on our proposed hazard function in Equation 4.7. It is clear that the estimated hazard rate fits the empirical hazard rate quite well, which means that our proposed hazard function is an appropriate one giving a good approximation to the real hazard rate. It can also be observed that the real hazard rate does decrease as the longer users are exposed to the tweet.

4.3 Model Implementation

In this section, we give detailed algorithms for model implementation. In particular, we show the parameter estimation algorithm and the cascade simulation algorithm.

4.3.1 Parameter Estimation

Given a cascade $\mathbb{C}(T)$, based on Equation 4.1, maximum likelihood estimation is used to estimate the unknown parameters,

$$\hat{\mathbf{\Theta}} = argmax_{\mathbf{\Theta}} log(P(\mathbb{C}(T); \mathbf{\Theta})).$$

The detailed algorithm is shown in Table 4.2.

The key part is to calculate the log-likelihood function $ll(\mathbf{\Theta}) = log(P(\mathbb{C}(T); \mathbf{\Theta}))$, then optimisation techniques such as gradient ascent can be used to estimate the best parameter $\hat{\mathbf{\Theta}}$. At each time step t (line 4), users who are just exposed to the information by time t are added into the set of type II Grey Node users $\mathbb{X}$ (line 5-7). For all the users in $\mathbb{X}$, the log-likelihood of their re-sharing

Input : social network G, cascade $\mathbb{C}(T)$, time interval dt, hazard function $h(t;\boldsymbol{\Theta})$

Output : estimated parameters $\hat{\boldsymbol{\Theta}}$

1: set log likelihood function $ll(\boldsymbol{\Theta}) = 0$
2: set type II users $\mathbb{X} = \{\}$
3: set $Current\_Sharers = \{u_0\}$
4: **for** $t = 0$ **to** T **step** dt :
5: **for** $u_i \in Current\_Sharers$
6: add $Follower^{(i)} \backslash \mathbb{C}(t)$ into set $\mathbb{X}$
7: **endfor**
8: **for** $u_i \in \mathbb{X}$
9: **if** $u_i \in \mathbb{C}(t, t+dt)$
10: $ll(\boldsymbol{\Theta}) = ll(\boldsymbol{\Theta}) + log(P_i(t;\boldsymbol{\Theta}))$
11: **else**
12: $ll(\boldsymbol{\Theta}) = ll(\boldsymbol{\Theta}) + log(1 - P_i(t;\boldsymbol{\Theta}))$
13: **endif**
14: **endfor**
15: **for** $u_i \in \mathbb{C}(t, t+dt)$
16: remove u_i from set $\mathbb{X}$
17: **endfor**
18: set $Current\_Sharers = \mathbb{C}(t, t+dt)$
19: **endfor**
20: $\hat{\boldsymbol{\Theta}} = argmax_{\boldsymbol{\Theta}} ll(\boldsymbol{\Theta})$
21: **return** $\hat{\boldsymbol{\Theta}}$

Table 4.2: The Parameter Estimation Algorithm.

behaviour at t is calculated (line 8-14). Finally, we have the log-likelihood function $ll(\boldsymbol{\Theta})$, and obtain the estimated parameters $\hat{\boldsymbol{\Theta}}$ by optimising it (line 20).

4.3.2 Cascade Simulation

Once the parameters $\boldsymbol{\Theta}$ are known, based on Equation 4.2, Monte Carlo simulation is used to simulate the cascade from time T_0 to time $T_0 + \Delta T$. The detailed algorithm is presented in Table 4.3.

In this algorithm, the key part is to simulate the re-sharing behaviour of each user at each time step t. Similar to the estimation algorithm, at each time step t (line 3), users who are just exposed to the information by time t are added into the set of type II Grey Node users $\mathbb{X}$ (line 4-6). For all the users

Input : social network G, cascade $\mathbb{C}(T_0)$ by time T_0, simulation duration ΔT,time interval dt,parameters $\mathbf{\Theta}$, hazard function $h(t;\mathbf{\Theta})$

Output : simulated cascade $\widetilde{\mathbb{C}}(T_0 + \Delta T)$

1: set type II users $\mathbb{X} = \{\}$
2: set $Current\_Sharers = \mathbb{C}(T_0)$
3: **for** $t = T_0$ **to** $T_0 + \Delta T$ **step** dt :
4: **for** $u_i \in Current\_Sharers$
5: add $Follower^{(i)} \backslash \mathbb{C}(t)$ into set $\mathbb{X}$
6: **endfor**
7: **for** $u_i \in \mathbb{X}$
8: draw a random number r from $U(0, 1)$
9: **if** $r < P_i(t;\mathbf{\Theta})$
10: add pair $< u_i, t + dt >$ into $\widetilde{\mathbb{C}}(t, t + dt)$
11: **endif**
12: **endfor**
13: **for** $u_i \in \widetilde{\mathbb{C}}(t, t + dt)$
14: remove u_i from set $\mathbb{X}$
15: **endfor**
16: set $Current\_Sharers = \widetilde{\mathbb{C}}(t, t + dt)$
17: **endfor**
18: **return** $\widetilde{\mathbb{C}}(T_0 + \Delta T)$

Table 4.3: The Cascade Simulation Algorithm.

in $\mathbb{X}$, their re-sharing behaviour at t are simulated (line 7-12). Finally at time $T_0 + \Delta T$, we have the simulated cascade $\widetilde{\mathbb{C}}(T_0 + \Delta T)$ (line 18).

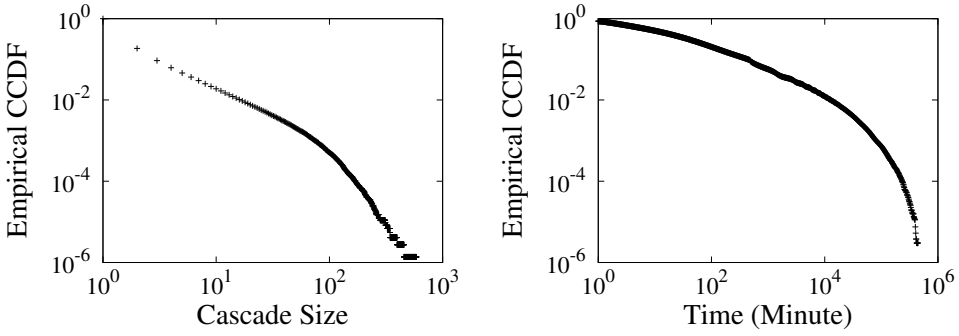
4.4 Experiment

4.4.1 Dataset

We use a Singapore based Twitter data set which contains more than 3 million users [96]. We crawl these users from a seed set of Singapore local celebrities and active users in a snowball-style way. The follow links between them and their tweets are periodically crawled. In this work, we use the subset of tweets from January 1st, 2010 to December 31th, 2012. From these tweets, we get all the retweets to construct retweeting cascades (see Figure 4.1). In all these cascades, we only consider the cascades in which the original tweet posters are

the Singapore users we crawled, so that we have the information about the root users of the cascades.

In all, we get 2,425,348 cascades which have at least one retweeter. Figure 4.4 (a) presents the cumulative distribution of the sizes of these cascades. It shows the large cascades are rare, which implies the difficulty in predicting the cascade growth, as most cascades do not grow anymore when they are in small sizes. Figure 4.4 (b) presents the cumulative distribution of the retweeting time delays of users after they are exposed to the tweets. From this figure, we can see even after long time, it is still possible for one tweet to be retweeted. This observation confirms our choice of hazard function in Equation 4.7, which is heavy tailed.



(a) (b)

Figure 4.4: (a) Cumulative distribution of the cascade sizes. (b) Cumulative distribution of the time delays.

4.4.2 Probabilistic Model Fitting

First, we evaluate the validity of our model. As the common way to check a probabilistic model, perplexity, which is a measurement of how well a probability model predicts a held-out sample, is used to measure the model validity. Given a set of cascades with size n, $\{\mathbb{C}_i(t)\}_{i=1}^n$, for each cascade $\mathbb{C}_i(t)$, we first observe it by time T_0, then ΔT later, based on a model $\mathcal{M}$, the probability $P_{\mathcal{M}}(\mathbb{C}_i(T_0 + \Delta T)|\mathbb{C}_i(T_0))$ is calculated. The formula of perplexity is defined in the Equation 4.9. The smaller the perplexity is, the better fitting the model

is.

$$Perplexity(\mathcal{M}) = e^{-\sum_{i=1}^{n} \frac{1}{n} \cdot log(P_{\mathcal{M}}(\mathbb{C}_i(T_0+\Delta T)|\mathbb{C}_i(T_0)))} \tag{4.9}$$

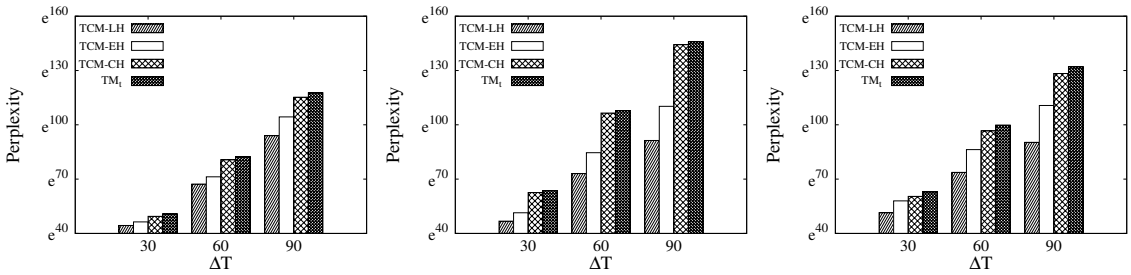
In this experiment, we examine: first whether the time-aware cascade model (TCM) is better than the traditional ones such as Threshold Model (TM) in terms of modelling information cascade over time; secondly whether our proposed hazard function in Equation 4.7 is more suitable in Twitter setting than others. As in TM, time is not a factor of concern, we adapt the models by projecting the growing cascade on the time dimension according to its original idea. In order to study how the different choices of hazard functions affect the time-aware model, we examine the following different choices of hazard functions – our proposed long tail hazard function (TCM-LH) in Equation 4.7, constant hazard function (TCM-CH) in Equation 4.12 and exponential hazard function (TCM-EH) in Equation 4.14. All these models are expressed in terms of hazard function, $h_i(t)$, which represents user u_i's re-sharing chance at time t. And all the parameters in these models are estimated by using maximum likelihood estimation algorithm listed in Table 4.2.

- **TM**$_t$: Threshold Model proposed the key concept "threshold", which in the setting of Twitter network is such a value: if the number of a user's followees who have retweeted one tweet exceeds this value, then this user retweets this tweet. In TM, the key point is threshold rather than time. In order to integrate time into this baseline TM$_t$, we use sigmoid function as the continuous thresholding function [32] and the hazard function is given in the following Equation 4.10.

$$h_i(t) = \lambda \cdot s(|Followee^{(i)}(t)|) \tag{4.10}$$

  where $s(x) = \frac{1}{1+e^{-a(x-b)}}$, and λ, a and b are parameters to be estimated.

- **TCM-CH**: Constant hazard in Equation 4.12 is the easiest way to define



(a) Year 2010 (b) Year 2011 (c) Year 2012

Figure 4.5: Perplexities of different models.

a hazard function, which is also considered in [39]. However, in this case $\lim_{\tau \to \infty} H(\tau) = \infty$, which does not satisfy the constraints listed in Table 4.1.

$$H(\tau) = \lambda \cdot \tau \tag{4.11}$$

$$h(\tau) = \frac{\mathrm{d}H(\tau)}{\mathrm{d}\tau} = \lambda \tag{4.12}$$

where parameter set $\mathbf{\Theta} = \{\lambda\}$.

- **TCM-EH**: As some works such as [60] reported, the exponential function may be a proper function to model the time delay of retweeting. In this baseline, we use the exponential hazard function in the following Equation 4.13 and 4.14, which also satisfy the constraints in Table 4.1.

$$H(\tau) = \lambda \cdot (1 - e^{-k \cdot \tau}) \tag{4.13}$$

$$h(\tau) = \frac{\mathrm{d}H(\tau)}{\mathrm{d}\tau} = \lambda \cdot k \cdot e^{-k \cdot \tau} \tag{4.14}$$

where parameter set $\mathbf{\Theta} = \{\lambda, k\}$.

In this experiment, T_0 is set to 30 minutes, and ΔT is set to 30, 60 and 90 minutes respectively. We evaluate these models on all the cascades which has a size larger than 10 at T_0 in year 2010, 2011 and 2012. Figure 4.5 shows the perplexities of different models. We can see for different year and ΔT the time-aware cascade models outperforms threshold model, and our model

TCM-LH performs best among these time-aware cascade models , which means TCM-LH is a proper model for growing retweeting cascades in Twitter. One interesting observation is that performances of different time-aware cascade models are very different, which implies that the choice of hazard function is critical for fitting model to real cascade data. The other observation is that the perplexities of all the models in year 2010 are much smaller than the perplexities in year 2011 and 2012, which is due to the smaller sizes of cascades in year 2010.

We also study the following two representative retweeting cascades: cascade (a) triggered by a tweet which promotes a music festival; cascade (b) triggered by a tweet about a local breaking news. For each cascade, each model learns the parameters and then simulates 100 cascades using the estimating algorithm and simulating algorithm in Section 4.3. Figure 4.6 shows the fitting performances of our proposed model (TCM-LH) and other baseline models for these two retweeting cascades (a) and (b). In each plot of Figure 4.6, the red curve shows how the number of users of the real cascade increases over time. The black curve shows the average number of users of the 100 simulated cascades over time, and the error bars around it are the standard deviations. The blue curve shows the simulated cascade which is nearest to the real cascade. We can see that: cascade (a) has a high initial retweeting rate, but its rate dramatically decreases; different from cascade (a), cascade (b) has a relatively slow initial retweeting rate, however, the tweet continually gets the interest of users and the cascade keeps growing over time. It can be observed that for both cascades our proposed model TCM-LH performs the best — the black and blue curves are close to the red curves. TCM-EH is worse than TCM-LH. Especially for cascade (b), TCM-EH can not generate a similar cascade to it. For other baseline models, it seems that they can not fit the real cascade well — the simulated cascades are far from the real cascades. Due to missing the effect of time in TCM-CH (constant hazard rate) and TM_t (threshold based

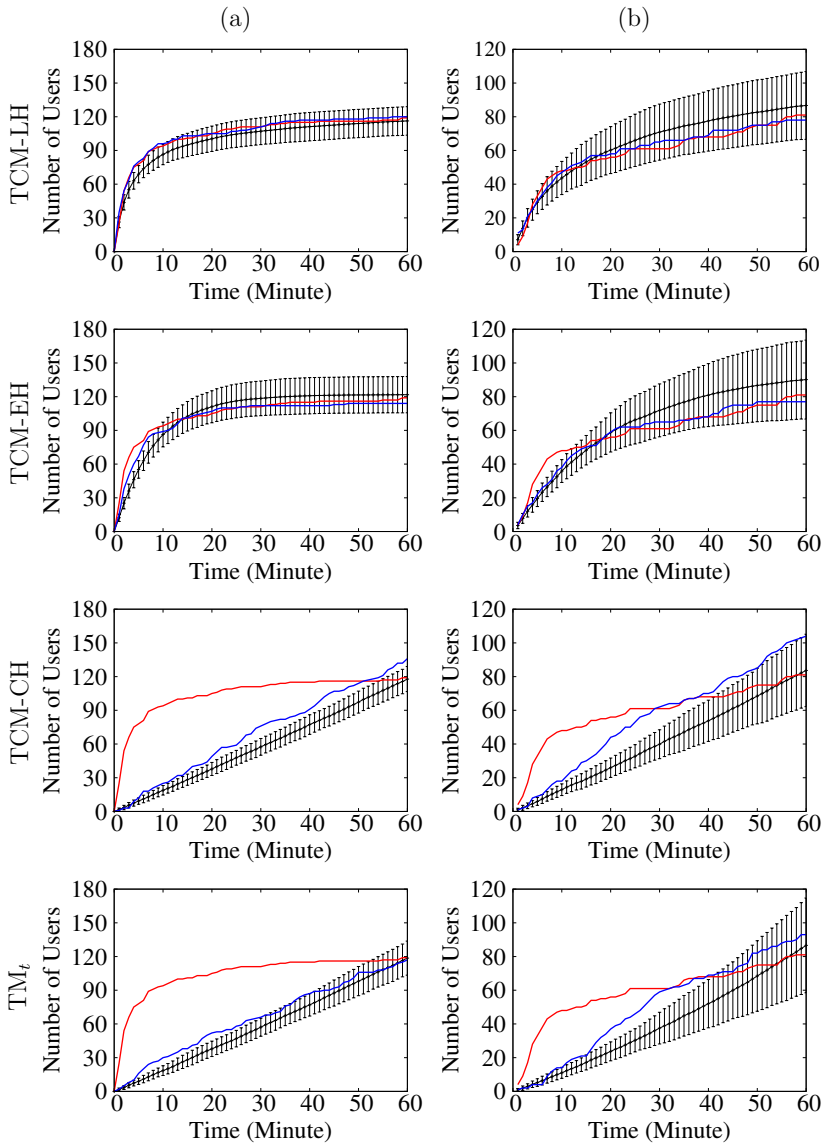


Figure 4.6: Fitting performances of our proposed model (TCM-LH) and other baseline models for two Twitter cascades (a) and (b).

hazard rate), the simulated cascades keep growing with roughly the same rates all the time. Another interesting observation is that the results of TCM-CH and TM$_t$ are similar. We examined these two cascades and found that for most retweeter, only one of their followees is in the cascade. It makes the learned threshold parameter b in TM$_t$ close to 1, so that there is no significant difference between TCM-CH and TM$_t$.

Besides, using our proposed model TCM-LH, the learned parameters for these two cascades are as follows: for cascade (a) $\lambda^{(a)} = 0.0082$, $\alpha^{(a)} = 1.35$, $\beta^{(a)} = 0.50$; for cascade (b) $\lambda^{(b)} = 0.0140$, $\alpha^{(b)} = 4.78$, $\beta^{(b)} = 0.45$. Although

cascade (a) has more retweeters at the early stage, based on the learned parameters above we make the following interpretations: $\lambda^{(b)} > \lambda^{(a)}$ means cascade (b) is more attractive in terms of the eventual retweeting probability; $\alpha^{(b)} > \alpha^{(a)}$ means the "life time" of cascade (b) is longer.

4.4.3 Predicting Cascade Growth

The other way to verify our proposed model is to evaluate its prediction performance. As mentioned in several existed works [17] [60], the initial information of a growing cascade in social networks can make the prediction much more accurate. It would be more practical to predict the cascade size after observing how the cascade grows initially, rather than to predict the cascade size from the very beginning. Here we conduct a prediction task, in which a cascade is tracked over time, and a sequence of predictions are made as the cascade grows. Rather than predict the final cascade size, each time we predict the cascade growth after a fixed time period (e.g. one hour), which is more practical.

Given n cascades $\{\mathbb{C}_i(t)\}_{i=1}^n$, the prediction time T_0, and fixed time period ΔT, denoting the growth of cascade $\mathbb{C}_i(t)$ as $\Delta_i(T_0, T_0+\Delta T) = |\mathbb{C}_i(T_0+\Delta T)| - |\mathbb{C}_i(T_0)|$, the following evaluation measures are considered:

- **Mean Absolute Error (MAE)**
  $MAE = \frac{\sum_{i=1}^{n} |\Delta_i(T_0,T_0+\Delta T) - \hat{\Delta}_i(T_0,T_0+\Delta T)|}{n}$

- **Relative Absolute Error (RAE)**
  $RAE = \frac{\sum_{i=1}^{n} |\Delta_i(T_0,T_0+\Delta T) - \hat{\Delta}_i(T_0,T_0+\Delta T)|}{\sum_{i=1}^{n} \Delta_i(T_0,T_0+\Delta T)}$

where $\hat{\Delta}_i(T_0, T_0 + \Delta T)$ is the estimation of $\Delta_i(T_0, T_0 + \Delta T)$.

Based on the algorithms in Section 4.3, TCM-LH predicts the cascade growth as follows. (I) Estimate the parameters based on $\mathbb{C}(T_0)$; (II) Simulate the cascade from time T_0 to $T_0 + \Delta T$ for a large number of times, then take the median size of simulated cascades at time $T_0 + \Delta T$.

We first compare TCM-LH with other baseline models in Section 5.4.2. We found that with big prediction errors, these baseline models are not suitable for this prediction task. We then compare TCM-LH to linear regression, which is widely applied in many prediction tasks such as popularity prediction in social media. We summarize most proposed factors [48] [6], which may drive the cascades grow or be relevant to the sizes of cascades in Twitter, including original tweeter features, tweet content features, social graph topological features and temporal features. And in the experiment, we found the temporal features are the most important predictors. These features of one tweet are denoted as a feature vector $\boldsymbol{f}$. We conduct the following two baselines.

- **LR1** In this baseline, the growth of a cascade is directly estimated. The regression formulation is given in the Equation 4.15. For some cascades, the increased cascade size is zero, so we use $log(\hat{\Delta}_i(T_0, T_0 + \Delta T) + 1)$ instead of $log(\hat{\Delta}_i(T_0, T_0 + \Delta T))$.

$$log(\hat{\Delta}_i(T_0, T_0 + \Delta T) + 1) = k_0 + \boldsymbol{k}^T \cdot \boldsymbol{f} \tag{4.15}$$

- **LR2** Some existing works such as [86] reported that there is a strong linear relationship found between the log-transformed popularities at different times. And $log(S(T_0))$ is indeed one feature included in the feature vector $\boldsymbol{f}$. So in this baseline, the cascade size at time $T_0 + \Delta T$, i.e. $S(T_0 + \Delta T)$, is first estimated. The regression formulation is given in the Equation 4.16.

$$\begin{aligned} log(\hat{S}(T_0 + \Delta T)) &= k_0 + \boldsymbol{k}^T \cdot \boldsymbol{f} \\ \hat{\Delta}_i(T_0, T_0 + \Delta T) &= \hat{S}(T_0 + \Delta T) - S(T_0) \end{aligned} \tag{4.16}$$

We randomly choose 10,000 cascades from our data set. These cascades are tracked over time (when T_0 = 5, 10, 15, 20, 25, 30 minutes), and at each time we predict the cascade growth one hour later (ΔT = 60 minutes).

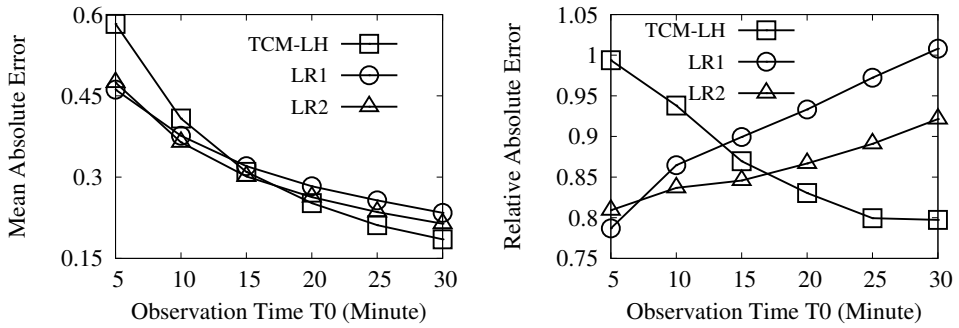


Figure 4.7: (a) Mean Absolute Error (MAE). (b)Relative Absolute Error (RAE).

10-fold cross validation is used. The prediction of TCM-LH is based on the median of 100 simulations. Figure 4.7 (a) and (b) show the Mean Absolute Error (MAE) and Relative Absolute Error (RAE) of TCM-LH, LR1 and LR2 respectively. We observe that: (I) In Figure 4.7 (a), the MAE decreases for all these three methods as longer we observe the cascades. One possible reason is that as time goes by, most cascades do not grow or only grow a little, so that MAE decreases over time. (II) However, in Figure 4.7 (b), we can see a very different trend: RAE does increase for LR1 and LR2 over time. One possible reason is that as time goes by, the correlation between the features and the cascade growth becomes smaller and smaller. (III) In Figure 4.7 (b), RAE does decrease for our model TCM-LH. Different from the feature based methods, TCM-LH models the retweeting process in Twitter network and predict the cascade growth based on the simulations of this process. So the longer we observe the cascades, the more accurate the estimations of the parameters in our model are, and the better the performance is.

4.4.4 Improving Virality Prediction Using TCM Simulation

Virality prediction at early stage is very useful for many applications such as viral marketing and breaking news detection. A straightforward method

Threshold	Measure	Random Guessing	Without Simulation	With Simulation
20	Recall	0.4817	0.4535	**0.6254**
	Precision	0.0034	**0.7285**	0.5678
	F1	0.0068	0.5590	**0.5952**
25	Recall	0.5764	0.4716	**0.5808**
	Precision	0.0026	**0.7500**	0.6215
	F1	0.0053	0.5791	**0.6005**
30	Recall	0.4600	0.4333	**0.5667**
	Precision	0.0014	**0.6915**	0.6071
	F1	0.0027	0.5328	**0.5862**
35	Recall	0.4653	0.3762	**0.5446**
	Precision	0.0009	**0.6909**	0.5612
	F1	0.0019	0.4872	**0.5528**
40	Recall	0.4545	0.2424	**0.4697**
	Precision	0.0006	**0.6667**	0.4247
	F1	0.0012	0.3556	**0.4460**

Table 4.4: The results of virality prediction for different solutions on different thresholds.

is learning the parameters of a cascade at early stage, and then predicting based on the simulations of TCM-LH (the same as Section 4.4.3). We found in practice this method doesn't work. Because TCM-LH only works when enough data is observed (as shown in Figure 4.7), but at the early stage, the sizes of most cascades are very small, which makes it hard to estimate the proper parameters of cascades at early stage. However, we can make use of the simulations of our model TCM-LH to improve vitality prediction by remedying the imbalance issue in cascade data.

In particular, we conduct the following virality prediction task: at time T_0 = 5 minutes, predict whether one cascade goes viral in the future, which means its cascade growth exceeds a prefixed threshold. The skewness of distribution of the cascade sizes (See Figure 4.4 (a)) is a challenge of this prediction task. In our data set, only 1‰ cascades grow over 35. Our TCM-LH model can be used to make this data set less skew by adding several simulated viral cascades into it. In this experiment, we randomly choose 100,000 cascades from our data set. 10-fold cross validation are used. There are two types of

training sets : (I) original training set without simulated viral cascades, (II) training set with simulated viral cascades: we choose the top 100 cascades from training set, then the parameters of these cascades are learned and each of them are simulated 100 times, and at last in all 10,000 simulated cascades are added into this training set. We use the features in Section 4.4.3 to learn the logistic regression classifier. Table 4.4 presents the prediction results of different solutions, from which we can observe that, after adding the simulated viral cascades into the training set, although the precision is not as good as before, the higher recall and F1 score are achieved. It shows that benefiting from the simulated viral cascades, the classifier can identify around 20% more viral cascades.

4.5 Summary

In this chapter, we use a general time-aware cascade model to describe the dynamic process of growing cascades in social networks over time. Based on this general model, a concrete model TCM-LH is designed for the retweeting cascades in Twitter. We conduct extensive evaluations based on a large real Twitter data set with over two million retweeting cascades. Our experiment results show our proposed TCM-LH fits the real cascade data better than other baselines in terms of model fitting. We also empirically show that our proposed TCM-LH could benefit applications such as virality prediction.

Chapter 5

Social Network Monitoring for Bursty Cascade Detection

In this chapter, we study how to select a budgeted set of social network sensors for busty cascade detection. We first formulate this problem as a constraint satisfaction problem which has high computational complexity. To reduce the computational cost, we then transform the problem into an LP (Linear Programming) problem. Furthermore, we use the sub-gradient method instead of the standard simplex method or interior-point method to solve the LP problem, which makes it possible for our solution to scale up to large social networks.

5.1 Problem Formulation

In a social network $G = \langle U, E \rangle$, where U and E represent the users and the social links between them respectively, we consider a set of cascades C developing on it. Each cascade $c \in C$ is a set of posts or tweets about the same topic. For example, in Twitter all the tweets which contain hashtag *#oscars2017* is a cascade. Specifically, each post or tweet d is represented as a pair $\langle d_u, d_t \rangle$, where d_u is its user and d_t is the corresponding timestamp. Further a cascade c is represented as a set of pairs, i.e. $c = \{\langle d_u, d_t \rangle\}$. Note that one user may join the same cascade many times, but with different timestamps. Figure 5.1

provides an example, i.e. $c = \{\langle u_1, t_1\rangle \langle u_2, t_2\rangle \langle u_1, t_3\rangle \langle u_3, t_4\rangle \langle u_4, t_5\rangle\}$.

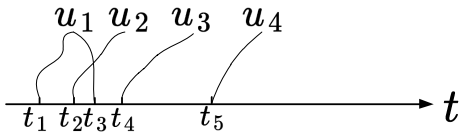


Figure 5.1: An example of cascade c.

As discussed in Section 2.2, there are two kinds of burst detection solutions: the ones which process data in an online way and the ones take a retrospective view of the data. Here we focus on the online burst detection solutions, which are more practical for early detection. Specifically, we generalize different online burst detection solutions as a classifier in the form $F_0 = \langle f, \delta_0\rangle$, where f is a function which measures the burstiness of c, i.e. $f : C \to \mathbb{R}$, and $\delta_0 \in \mathbb{R}$ is some threshold. F_0 works as follows: $F_0(c) = 1$, if $f(c) \geq \delta_0$, otherwise, $F_0(c) = 0$. One naive example is a classifier which monitors the size of a cascade, and reports it as a burst if its size reaches some predefined threshold. For example, if the number of tweets which mention *#oscars2017* exceeds a predefined threshold, say one hundred thousand, it is identified as a burst. In this case, f is simply the size of a cascade c, i.e. $f(c) = |c|$. Other examples include arrival rate, i.e. the number of tweets per minute or hour, the significance score proposed in [82] and the acceleration defined in [44, 97].

In this work, we do not intend to create a new solution for burst detection. Instead, we try to find a good way to apply existing detection solutions under the constraint of limited resources. More specifically, suppose we can only follow budgeted users, say m users instead of the whole population U. This means we have to choose a subset S from U where $|S| = m$. Therefore for each cascade c, what we can observe is a "shrunken" cascade $c_S = \{\langle d_u, d_t\rangle \in c | d_u \in S\}$. Denote $C_S = \{c_S | c \in C\}$. The problem is, given a classifier F_0, how to select a subset $S \subset U$ ($|S| = m$) so that $F_0(c_S) = F_0(c)$ for every $c \in C$. Ideally our goal is to solve the following constraint satisfaction problem.

Table 5.1: Summary of notations used in this chapter.

Notation	Description
U	all the social network sensors (or users)
S	a subset of social network sensors
d	a post or tweet
d_u, d_t	the user and timestamp of d
c	a cascade
C	a set of cascades
c_S	a sub-cascade observed only from S
C_S	a set of sub-cascades observed only from S
c_u	a sub-cascade observed only from user u
m	the number of sensors (budget)
f	a function measuring the burstiness of c
F_0, F_1	the classifiers
δ_0, δ_1	the thresholds
$c(t)$	a cascade observed by time t
C_{burst}	a set of bursty cascades
DT_c	detection time of cascade c
Y_c	a coefficient vector defined by $\{f(c_u(DT_c))\}_{u \in U}$

Given a set of cascades C and a classifier $F_0 = \langle f, \delta_0 \rangle$,

$$\begin{aligned} \text{find} \quad & S \subset U \\ \text{subject to} \quad & f(c_S) \geq \delta_0, \forall c \in \{c \in C | f(c) \geq \delta_0\} \\ & f(c_S) < \delta_0, \forall c \in \{c \in C | f(c) < \delta_0\} \\ & |S| = m. \end{aligned} \tag{5.1}$$

Note that, we assume that the given classifier F_0 can achieve the detection performance we desire. And as shown in Problem 5.1 our task is to select a budgeted set S such that F_0 works on C_S as well as on C.

Table 5.1 summarizes the notations used in this chapter.

5.2 LP Model

Considering there are millions of cascades and users in social networks, which leads to a large number of constraints and variables (i.e. $|C|$ constraints and $|U|$ variables), the complexity of the given constraint satisfaction problem could

be high [22]. In this section, we relax the constraints in Problem 5.1, and transform it into an LP (Linear Programming) problem which can be solved more efficiently.

5.2.1 Constraint Reduction

In the first step, we reduce most of the constraints in Problem 5.1.

It is straightforward that, a feasible solution may not exist in Problem 5.1. For instance, consider $f = |c|$, then $f(c_S) \leq f(c)$. In an extreme case, where $\delta_0 > m$, we have $f(c_S) \leq f(c) = m < \delta_0$ for all the cascades. This means no burst can be detected on C_S for any $S \subset U$.

Intuitively, applying the same threshold δ_0 on C_S may cause lower recall. So it is reasonable to use another slightly different classifier $F_1 = \langle f, \delta_1 \rangle$, where δ_1 can be different from δ_0, instead of $F_0 = \langle f, \delta_0 \rangle$. It gives the following Problem 5.2.

Given a set of cascades C and a classifier $F_0 = \langle f, \delta_0 \rangle$,

$$\begin{aligned} \text{find} \quad & S \subset U, \delta_1 \in \mathbb{R} \\ \text{subject to} \quad & f(c_S) \geq \delta_1, \forall c \in \{c \in C | f(c) \geq \delta_0\} \\ & f(c_S) < \delta_1, \forall c \in \{c \in C | f(c) < \delta_0\} \\ & |S| = m. \end{aligned} \tag{5.2}$$

Even so, there would still be millions of cascades, which means we have millions of constraints to handle. It may be intractable to find a feasible solution which satisfies so many constraints. Fortunately, we can reduce most of these constraints by observing that there are actually very few positive cases, i.e. bursty cascades, among the whole population of cascades. (Figure 5.5 in the experiment part shows that the large cascades are rare.) So we just focus on the positive cases, and ignore all the negative cases. At the same time, we set the threshold δ_1 as high as possible. In this way, we can capture all

the bursty cascades, and at the same time filter out the non-bursty cascades as many as possible. Therefore, we transform Problem 5.2 into the following Problem 5.3.

Given a set of cascades C and a classifier $F_0 = \langle f, \delta_0 \rangle$,

$$\begin{aligned} \underset{S,\delta_1}{\text{maximize}} \quad & \delta_1 \\ \text{subject to} \quad & f(c_S) \geq \delta_1, \forall c \in \{c \in C | f(c) \geq \delta_0\} \\ & |S| = m. \end{aligned} \tag{5.3}$$

In our experiment, we found that in this step the number of constraints can be significantly reduced, i.e. from millions of constraints to thousands of constraints. Notice that, for Problem 5.3 a feasible solution always exists because just set $\delta_1 = -\infty$, all the constraints in Problem 5.3 must be satisfied.

5.2.2 Linear Transformation

Although having much fewer constraints, it is still hard to design a general algorithm for Problem 5.3 because the concrete formulas of f are different for various burst detection solutions. Fortunately, we found that for some specific classifiers, e.g. [44, 97], the burstiness evaluating function f is *additive*. We will show that by benefiting from this property we can transform all the constraints in Problem 5.3 into *linear* constraints. From now on, we focus on the cases in which f is additive, and later we will discuss the cases in which f is a non-additive function in Section 5.2.5.

Formally, f is additive on the power set of U, i.e. 2^U, if for any c and $S = S_1 \cup S_2$, where $c \in C, S, S_1, S_2 \subset U$ and $S_1 \cap S_2 = \emptyset$, f satisfies the property of additive separability:

$$f(c_S) = f(c_{S_1}) + f(c_{S_2})$$

There are many such examples. A trivial example is the size function

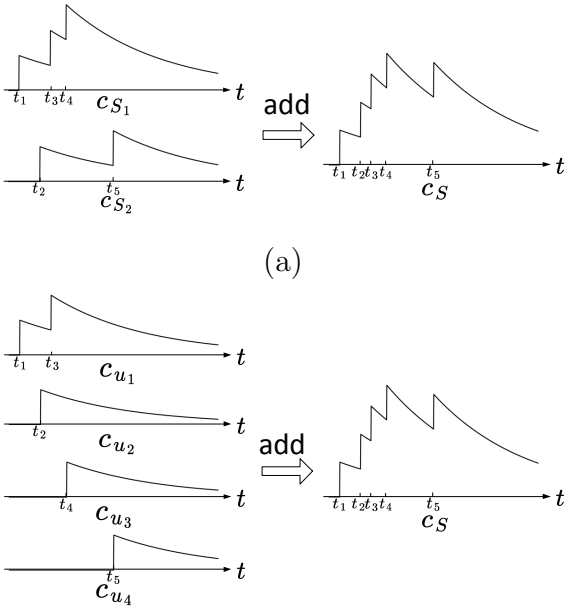
$f = |c|$. It is obvious that $|c_S| = |c_{S_1}| + |c_{S_2}|$, if $S_1 \cap S_2 = \emptyset$. Another example is the arrival rate, i.e. the number of tweets per minute or hour, which is an intuitive measure of burstiness. Other important examples include the *velocity* and *acceleration* defined in [97], and *MACD* defined in [44]. The linearity of *MACD* is proved in [44]. Here we will show that *velocity* and *acceleration* are additive. Later, in the experiment part, we will adopt acceleration as one of the burstiness evaluating functions.

Adapting the definition in [97], we define the *velocity* $\hat{v}$ and *acceleration* $\hat{a}$ of a cascade $c \in C$ as follows.

$$\hat{v}_{\Delta T}(t) = \sum_{\langle d_u, d_t \rangle \in c \wedge d_t \leq t} \frac{exp((d_t - t)/\Delta T)}{\Delta T}$$
$$\hat{a}_c(t) = \frac{\hat{v}_{\Delta T_2}(t) - \hat{v}_{\Delta T_1}(t)}{\Delta T_1 - \Delta T_2}$$

The exponential part in $\hat{v}_{\Delta T}(t)$ works like a soft moving window, which gives the recent terms high weight, but gives low weight to the ones far away, and the smoothing parameter ΔT is the window size. To capture the change of velocity, acceleration $\hat{a}_c(t)$ is defined as the difference of velocities with different window size ΔT_1 and ΔT_2. (A real example is presented in Figure 5.12 in the experiment part.) It is clear that as a sum of weighted items, $\hat{v}$ is additive. In turn, as a linear combination of $\hat{v}$, $\hat{a}$ is also additive. Consider the cascade c in Figure 5.1, $S_1 = \{u_1, u_3\}$, $S_2 = \{u_2, u_4\}$ and $S = S_1 \cup S_2$, Figure 5.2 (a) illustrates that velocity $\hat{v}$ is additive. Note that velocity $\hat{v}$ and acceleration $\hat{a}$ are actually functions of time. To measure the burstiness of cascade c, we mean $\hat{v}$ or $\hat{a}$ at a particular time point.

The advantage of additive functions lies in the fact that we can easily calculate any $f(c_S)$ by the strategy of *divide and conquer*. Specifically, for each cascade c, we can "divide" it as $\bigcup_{u \in U} c_{\{u\}}$, where $c_{\{u\}}$ is the sub-cascade observed only from user u. (For instance, $c_{\{u_1\}} = \{\langle u_1, t_1 \rangle, \langle u_1, t_3 \rangle\}$.) In the



(a)

(b)

Figure 5.2: (a) Velocity $\hat{v}$ is additive. (b) Each additive function can be "broken down" to "user" level.

similar way, we have $c_S = \bigcup_{u \in S} c_{\{u\}}$. For the sake of simplicity, we denote $c_{\{u\}}$ as c_u. As for any two different users u_1 and u_2, $c_{u_1} \cap c_{u_2} = \emptyset$, for any additive function f, we have

$$f(c_S) = f(\bigcup_{u \in S} c_u) = \sum_{u \in S} f(c_u)$$

This equation is demonstrated in Figure 5.2 (b). It means each additive function can be "broken down" to "user" level.

Therefore, we can transform the constraints in Problem 5.3 into linear constraints by simply replacing $f(c_S)$ with $\sum_{u \in S} f(c_u)$. It leads to the following Problem 5.4.

Given a set of cascades C and a classifier $F_0 = \langle f, \delta_0 \rangle$,

$$\begin{aligned} \underset{S,\delta_1}{\text{maximize}} \quad & \delta_1 \\ \text{subject to} \quad & \sum_{u \in S} f(c_u) \geq \delta_1, \forall c \in \{c \in C | f(c) \geq \delta_0\} \\ & |S| = m. \end{aligned} \tag{5.4}$$

For each cascade c and each user u, we can calculate $f(c_u)$ beforehand, so that we can treat it as a constant. For the example illustrated in Figure 5.1, if set $f = |c|$, we have $f(c_{u_1}) = 2$, $f(c_{u_2}) = 1$ and so on. Further, denote x_u as the indicator of user u, X as the corresponding vector, i.e. $X = \{x_u\}$. It leads to the following 0-1 linear programming problem[1].

Given a set of cascades C and a classifier $F_0 = \langle f, \delta_0 \rangle$,

$$\begin{aligned} \underset{X,\delta_1}{\text{maximize}} \quad & \delta_1 \\ \text{subject to} \quad & \sum_{u \in U} x_u \cdot f(c_u) \geq \delta_1, \forall c \in \{c \in C | f(c) \geq \delta_0\} \\ & x_u \in \{0, 1\}, \forall u \in U \\ & \sum_{u \in U} x_u = m. \end{aligned} \tag{5.5}$$

5.2.3 Linear Programming Relaxation

In general, 0-1 linear program is NP-hard, which means it is hard to solve Problem 5.5 in large scale networks. A common way is to relax it to a linear program which is solvable in polynomial time. By replacing the constraint that $x_u \in \{0, 1\}$ by a weaker constraint that $x_u \in [0, 1]$, we have the following LP Problem 5.6.

Given a set of cascades C and a classifier $F_0 = \langle f, \delta_0 \rangle$,

$$\begin{aligned} \underset{X,\delta_1}{\text{maximize}} \quad & \delta_1 \\ \text{subject to} \quad & \sum_{u \in U} x_u \cdot f(c_u) \geq \delta_1, \forall c \in \{c \in C | f(c) \geq \delta_0\} \\ & 0 \leq x_u \leq 1, \forall u \in U \\ & \sum_{u \in U} x_u = m. \end{aligned} \tag{5.6}$$

As the solution of Problem 5.6, x_u can be interpreted as the probability that we choose user u into the subset S. In this way, $\mathbb{E}_X[f(c_S)] = \sum_{u \in U} x_u \cdot f(c_u)$

[1]Strictly, it is a mixed integer programming problem because δ_1 is a real value.

and $\mathbb{E}_X[|S|] = \sum_{u \in U} x_u$. It means, if choose user u with probability x_u, we can guarantee that $\mathbb{E}[f(c_S)] \geq \delta_1, \forall c \in \{c \in C | f(c) \geq \delta_0\}$ and $\mathbb{E}[|S|] = m$. To avoid uncertainty, we apply a simple greedy strategy: choosing m users with the largest probability x_u.

5.2.4 Detection Time

In previous sections, for the sake of simplicity, we had not considered the detection time. One problem of this is that a burst may be detected with a very long detection delay. And it is less useful to detect an event if this event has already happened for a long time. Take the cascade in Figure 5.1 as an example, and suppose we have burstiness evaluation function $f(c) = |c|$ and budget $m = 2$. If we do not consider the detection time, solution $S_1 = \{u_1, u_2\}$ is the same as solution $S_2 = \{u_1, u_4\}$ because $f(c_{S_1}) = f(c_{S_2}) = 3$. However, the first time when c_{S_1} reaches the size of 3 is t_3, while it is t_5 for c_{S_2}. In real-world applications, we prefer S_1 which has an earlier detection time. It motivates us to integrate the detection time into the LP problem.

Denote $c(t) = \{\langle d_u, d_t \rangle \in c | d_t \leq t\}$ as a cascade observed by time t. The set of bursty cascades is defined as $C_{burst} = \{c \in C | \max_t\{f(c(t))\} \geq \delta_0\}$. For each $c \in C_{burst}$, denote $DT_c = \min\{t | f(c(t)) \geq \delta_0\}$ as the detection time, i.e. the first time identifying cascade c as a bursty cascade (See the example in Figure 5.3). We can transform the first constraint in Problem 5.6 as follows.

$$\max_t\{f(c_S(t))\} = \max_t\{\sum_{u \in U} x_u \cdot f(c_u(t))\} \geq \delta_1, \forall c \in C_{burst}$$

We add the constraint $DT_{c_S} \leq DT_c, \forall c \in C_{burst}$, i.e. the detection time for the sub-cascade c_S must be at least as early as the detection time for the original cascade c. It gives us the following constraint.

$$\max_{t \leq DT_c}\{\sum_{u \in U} x_u \cdot f(c_u(t))\} \geq \delta_1, \forall c \in C_{burst}$$

For specific evaluation functions, such as $f(c) = |c|$, we have $f(c(t_2)) \geq f(c(t_1))$, if $t_2 \geq t_1$ because cascade always grows. So we have,

$$\max_{t \leq DT_c} \{\sum_{u \in U} x_u \cdot f(c_u(t))\} = \sum_{u \in U} x_u \cdot f(c_u(DT_c))$$

Although for most other evaluation functions, this equation does not hold, we still can use $\sum_{u \in U} x_u \cdot f(c_u(DT_c))$ to replace $\max_{t \leq DT_c} \{\sum_{u \in U} x_u \cdot f(c_u(t))\}$. Because $\max_{t \leq DT_c} \{\sum_{u \in U} x_u \cdot f(c_u(t))\} \geq \sum_{u \in U} x_u \cdot f(c_u(DT_c))$, we have

$$\sum_{u \in U} x_u \cdot f(c_u(DT_c)) \geq \delta_1 \implies \max_{t \leq DT_c} \{\sum_{u \in U} x_u \cdot f(c_u(t))\} \geq \delta_1$$

In other words, the constraint $\sum_{u \in U} x_u \cdot f(c_u(DT_c)) \geq \delta_1$ guarantees $DT_{c_S} \leq DT_c$.

So adding the constraint of detection time, we transform Problem 5.6 into the following Problem 5.7.

$$\begin{aligned} \underset{X, \delta_1}{\text{maximize}} \quad & \delta_1 \\ \text{subject to} \quad & \sum_{u \in U} x_u \cdot f(c_u(DT_c)) \geq \delta_1, \forall c \in C_{burst} \\ & 0 \leq x_u \leq 1, \forall u \in U \\ & \sum_{u \in U} x_u = m. \end{aligned} \tag{5.7}$$

Here we give an example to demonstrate how to construct the LP problem from a set of cascades. Figure 5.3 shows a cascade set $C = \{c_1, c_2, c_3\}$ from a user set $U = \{u_1, u_2, u_3, u_4\}$. Suppose the threshold $\delta_0 = 5$, the budget $m = 2$ and use $f(c) = |c|$ as the burstiness evaluation function. Note that here we adopt cascade size just for illustration. In the experiment part, more sophisticated burstiness evaluation functions are tested, including the acceleration proposed in Chapter 3.

Based on the given setting, we have bursty cascade set $C_{burst} = \{c_1, c_2\}$.

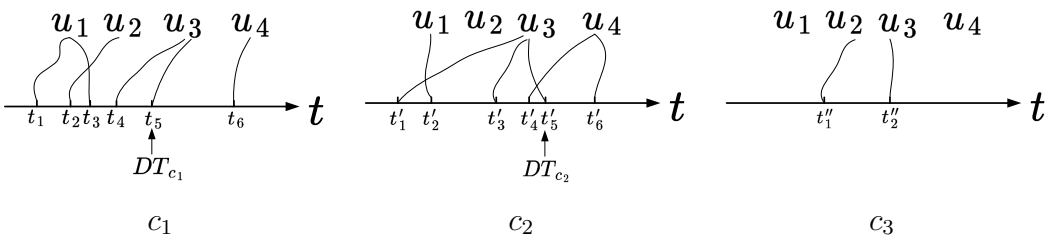


Figure 5.3: A toy example: cascades c_1 c_2 and c_3.

And for each bursty cascade, the detection time $DT_{c_1} = t_5$, $DT_{c_2} = t'_5$. For cascade c_1, by detection time t_5, we have sub-cascades $c_{u_1}(t_5) = \{\langle u_1, t_1\rangle \langle u_1, t_3\rangle\}$, $c_{u_2}(t_5) = \{\langle u_2, t_2\rangle\}, c_{u_3}(t_5) = \{\langle u_3, t_4\rangle \langle u_3, t_5\rangle\}, c_{u_4}(t_5) = \emptyset$, and their sizes are $2, 1, 2, 0$ respectively. Similarly for cascade c_2, by detection time t'_5, the sizes of sub-cascades are $1, 0, 3, 1$ respectively. For the sake of simplicity, denote x_{u_i} as x_i. We can construct the following LP problem from the example in Figure 5.3.

$$
\begin{aligned}
\underset{x_1,x_2,x_3,x_4,\delta_1}{\text{maximize}} \quad & \delta_1 \\
\text{subject to} \quad & 2x_1 + 1x_2 + 2x_3 + 0x_4 \geq \delta_1 \text{ (for cascade } c_1\text{)} \\
& 1x_1 + 0x_2 + 3x_3 + 1x_4 \geq \delta_1 \text{ (for cascade } c_2\text{)} \\
& 0 \leq x_i \leq 1, \forall 1 \leq i \leq 4 \text{ (boundary)} \\
& \sum_{i=1}^{4} x_i = 2 \text{ (budget limitation)}
\end{aligned}
$$

The solution of this LP problem is $\{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, \delta_1 = 4\}$, which means the optimal subset is $S = \{u_1, u_3\}$.

5.2.5 Non-additive Function

In this section, we consider the cases in which f is non-additive. As discussed above, non-additive f leads to intractable non-linear constraints. One way is to use another additive function $f^\star$ instead of the original f in the constraints. It leads to the following Problem 5.8.

Given a set of cascades C and a classifier $F_0 = \langle f, \delta_0 \rangle$, where f is a non-additive,

$$\begin{aligned} \underset{X,\delta_1}{\text{maximize}} \quad & \delta_1 \\ \text{subject to} \quad & \sum_{u \in U} x_u \cdot f^\star(c_u(DT_c)) \geq \delta_1, \forall c \in C_{burst} \\ & 0 \leq x_u \leq 1, \forall u \in U \\ & \sum_{u \in U} x_u = m. \end{aligned} \tag{5.8}$$

where $f^\star$ is an additive function. Recall $DT_c = \min\{t | f(c(t)) \geq \delta_0\}$ and $C_{burst} = \{c \in C | \max_t\{f(c(t))\} \geq \delta_0\}$. The original function f is actually used for identifying the bursty cascades.

The underlying logic here is that the cascades identified by different burst detection classifiers are similar. Actually most of these cascades share the same features: they are large cascades; they have big peaks and so on. For bursty cascades, it is reasonable to expect that $f^\star(c_S(t)) \approx f(c_S(t))$ in some way. The experiment in Section 5.4 shows that this replacement works on real data.

5.3 Scaling Up The Solution

To solve a general linear programming problem, the standard solutions are the simplex method [21] and the interior-point method [10]. Considering the large number of variables, (i.e. $|U|$ the number of users in a social network), these methods are not scalable to large networks.

Fortunately, we found that Problem 5.7 is actually equivalent to a convex optimization problem. This provides us with an opportunity to develop a more efficient algorithm to solve the LP problem.

Recall $C_{burst} = \{c \in C | \max_t\{f(c(t))\} \geq \delta_0\}$, which is the set of bursty cascades. Denote the coefficient vector in Problem 5.7 as $Y_c = \{f(c_u(DT_c))\}_{u \in U}$. Denote boundary $B = \{X | 0 \leq x_u \leq 1, \forall u \in U \wedge \sum_{u \in U} x_u = m\}$, which represents the constraints in Problem 5.7. We have the following Lemma 5.3.1 and

5.3.2.

Lemma 5.3.1. *Problem 5.7 is equivalent to following Problem 5.9.*

$$\underset{X}{minimize} \quad \left\{ \max_{c \in C_{burst}} \{-Y_c^\top X\} + I_B(X) \right\} \tag{5.9}$$

where $I_B(X)$ is the indicator function of B, i.e.

$$I_B(X) = \begin{cases} 0 & \text{if } X \in B \\ \infty & \text{otherwise} \end{cases}$$

Proof. Problem 5.7 $\implies$ Problem 5.9.

Notice that for Problem 5.7 a feasible solution always exists because $B \neq \emptyset$, as long as we set $\delta_1 = -\infty$, all the constraints in Problem 5.7 must be satisfied. And B is a closed set, so an optimal solution always exists. Denote $X^\star, \delta_1^\star$ as an optimal solution of Problem 5.7. We first show: $X^\star$ is also an optimal solution of Problem 5.9, i.e. $\max_{c \in C_{burst}} \{-Y_c^\top X^\star\} + I_B(X^\star) \leq \max_{c \in C_{burst}} \{-Y_c^\top X\} + I_B(X), \forall X$. Actually, because $X^\star$ satisfies all the constraints in Problem 5.7, $X^\star \in B$, i.e. $I_B(X^\star) = 0$. And we have the constraints $Y_c^\top X^\star \geq \delta_1^\star, \forall c \in C_{burst} \implies -Y_c^\top X^\star \leq -\delta_1^\star, \forall c \in C_{burst} \implies -\delta_1^\star \geq \max_{c \in C_{burst}} \{-Y_c^\top X^\star\}$. If $X \notin B$, $I_B(X) = \infty$, we have $\max_{c \in C_{burst}} \{-Y_c^\top X^\star\} + I_B(X^\star) \leq \max_{c \in C_{burst}} \{-Y_c^\top X\} + I_B(X) = \infty$. If $X \in B$, and suppose $\max_{c \in C_{burst}} \{-Y_c^\top X^\star\} + I_B(X^\star) > \max_{c \in C_{burst}} \{-Y_c^\top X\} + I_B(X)$ $\implies \max_{c \in C_{burst}} \{-Y_c^\top X^\star\} > \max_{c \in C_{burst}} \{-Y_c^\top X\} \implies -\delta_1^\star \geq \max_{c \in C_{burst}} \{-Y_c^\top X^\star\} > \max_{c \in C_{burst}} \{-Y_c^\top X\} = -\min_{c \in C_{burst}} \{Y_c^\top X\} \implies \delta_1^\star < \min_{c \in C_{burst}} \{Y_c^\top X\}$. It is obvious that $X, \delta_1 = \min_{c \in C_{burst}} \{Y_c^\top X\}$ is a feasible solution of Problem 5.7. But $\delta_1 > \delta_1^\star$. This contradicts the assumption that $X^\star, \delta_1^\star$ as an optimal solution of Problem 5.7. So $X^\star$ is also an optimal solution of Problem 5.9.

Problem 5.9 $\implies$ Problem 5.7.

Denote $X^\star$ as the optimal solution of Problem 5.9, i.e. $\max_{c \in C_{burst}} \{-Y_c^\top X^\star\} + I_B(X^\star) \leq \max_{c \in C_{burst}} \{-Y_c^\top X\} + I_B(X), \forall X$. We show that: $X^\star, \delta_1^\star = \min_{c \in C_{burst}} \{Y_c^\top X^\star\}$

is also an optimal solution of Problem 5.7. First, it is obvious that $X^\star \in B$ because $I_B(X^\star)$ is bounded. This implies that $\max_{c \in C_{burst}} \{-Y_c^\top X^\star\} \leq \max_{c \in C_{burst}} \{-Y_c^\top X\}, \forall X \in B$. It is also straight forward that $\delta_1^\star = \min_{c \in C_{burst}} \{Y_c^\top X^\star\} \leq Y_c^\top X^\star, \forall c \in C_{burst}$. So $X^\star, \delta_1^\star$ satisfies all the constraints in Problem 5.7. Suppose X, δ_1 is a feasible solution of Problem 5.7. We can see that $\delta_1^\star = \min_{c \in C_{burst}} \{Y_c^\top X^\star\} = -\max_{c \in C_{burst}} \{-Y_c^\top X^\star\} \geq -\max_{c \in C_{burst}} \{-Y_c^\top X\} = \min_{c \in C_{burst}} \{Y_c^\top X\} \geq \delta_1$. So $X^\star, \delta_1^\star$ is an optimal solution of Problem 5.7. □

Besides the equivalence between Problem 5.7 and Problem 5.9, Lemma 5.3.1 also tells us the optimal threshold of Problem 5.7 is the lower bound of $\{Y_c^\top X\}$, i.e.

$$\delta_1 = \min_{c \in C_{burst}} \{Y_c^\top X\}$$

Lemma 5.3.2. *Problem 5.9 is a convex optimization problem.*

Proof. Notice that as an affine function, $-Y_c^\top X$ is convex. Because the maximum of convex functions is also convex [10], $\max_{c \in C_{burst}} \{-Y_c^\top X\}$ is convex. And as the box B is a convex set, the indicator function $I_B(X)$ is also convex. It shows that the objective function $\max_{c \in C_{burst}} \{-Y_c^\top X\} + I_B(X)$ is convex. Therefore, Problem 5.9 is a convex optimization problem. □

Usually gradient descent method is used to solve a convex problem. However here in Problem 5.9 $\max_{c \in C_{burst}} \{-Y_c^\top X\}$ is not differentiable. So we use sub-gradient method [75] to solve it. Similar to gradient method, in sub-gradient method, the key is to calculate the moving direction, i.e. the sub-gradient, at each step. Because at point X, $-Y_{c^\star}$ is in the sub-gradient set, where $c^\star = \text{argmin}_{c \in C_{burst}} \{Y_c^\top X\}$, at each step we move along $Y_{c^\star}$. For the step size, backtracking search is not suitable for sub-gradient method. We consider diminishing the step size, which is a common step-size rule. Besides, as X is limited in B, the projected gradient method is applied to make sure at each

Algorithm 4: Sub-gradient Method

Input: Y_c: the burstiness score vector.
Input: K: the maximum number of iterations.
Input: γ: the initial step size.
Input: ϵ: threshold.
Output: X.

```
1  Initialize X ∈ B;
2  for k = 1 to K do
3      c* = argmin_{c∈C_burst} {Y_c^⊤ X};
4      grad = −Y_{c*};
5      step = γ/k;
6      X_new = X − step · grad;
7      X_new = P_B(X_new);
8      if |X_new − X| < ε then
9          X = X_new;
10         break;
11     end
12     X = X_new;
13 end
14 return X.
```

step X remains in B. The projection operator is presented in Algorithm 5. In line 2-9, X is projected into the box $\{X|0 \leq x_u \leq 1, \forall u \in U\}$. In line 10, X is projected on the hyperplane $\{X|\sum_{u\in U} x_u = m\}$.

The whole procedure is presented in Algorithm 4. In line 3-4, we calculate the sub-gradient. In line 5, the step size is divided by the iteration number k (the diminishing step-size rule). In line 6, we update current point X_{new} according to the sub-gradient. In line 7, X is projected into B. In line 8-11, we check whether the algorithm converges. Lemma 5.3.2 guarantees the convergence of Algorithm 4.

Interestingly, it can be observed that Algorithm 4 is actually reasonable and understandable in the sense that, in each iteration the algorithm finds the lower bound of the burstiness scores of C_{burst} (i.e. $\{Y_c^\top X\}$, see line 3 in Algorithm 4) and tries to push up this lower bound, which is actually the threshold δ_1 in Problem 5.7 (See Figure 5.4).

Besides, the algorithm's computational complexity is $O(|U| \cdot |C_{burst}| \cdot K)$. Because in practice Y_c is sparse, the actual computational cost is far less than

Algorithm 5: Projection Operator P_B

Input: X : the input vector.
Output: $X^\star$: the vector after projection.

$X^\star = X$;
for $u \in U$ **do**
  if $X^\star_u > 1$ **then**
    $X^\star_u = 1$;
  end
  if $X^\star_u < 0$ **then**
    $X^\star_u = 0$;
  end
end
$X^\star = X^\star - \frac{1^\top X^\star}{|U|} \cdot 1 + \frac{m}{|U|}$;
return $X^\star$

this theoretical cost. More importantly, the greedy strategy based solutions, such as [64, 106], select sensors one by one. Their computational costs are proportional to m, i.e. the number of chosen sensors. In contrast, the computational cost of our algorithm is invariant to m.

In the rest of this section, we take the cascades in Figure 5.3 as an example to illustrate Algorithm 4. First, we have following coefficient vectors for bursty cascade $\{c_1, c_2\}$.

$$Y_1 = (2, 1, 2, 0)^\top, Y_2 = (1, 0, 3, 1)^\top$$

We initialize X as $\frac{m}{n} = 0.5$, i.e. $X = (0.5, 0.5, 0.5, 0.5)^\top$, and set initial step size $\gamma = 0.25$.

- Iteration 1.

  Diminish step size: $step = \frac{\gamma}{1} = 0.25$.

  Calculate sub-gradient: $Y_1^\top X = 2.5, Y_2^\top X = 2.5, Y_1 <= Y_2 \implies grad = -Y_1, \delta_1 = 2.5$.

  Update X: $X = X - \gamma * grad = X + 0.25 \cdot Y_1 = (1.0, 0.75, 1.0, 0.5)^\top$.

  Project X into the unit cube: X is already in the unit cube.

  Project X on the hyperplane: $X = (0.6875, 0.4375, 0.6875, 0.1875)^\top$.

- Iteration 2.

  Diminish step size: $step = \frac{\gamma}{2} = 0.125$.

  Calculate sub-gradient:

  $Y_1^\top X = 3.1875, Y_2^\top X = 2.9375, Y_1 > Y_2 \implies grad = -Y_2, \delta_1 = 2.9375$.

  Update X: $X = X - \gamma * grad = X + 0.125 \cdot Y_2 = (0.8125, 0.4375, 1.0625, 0.3125)^\top$.

  Project X into the unit cube: $X = (0.8125, 0.4375, 1.0, 0.3125)^\top$.

  Project X on the hyperplane: $X = (0.671875, 0.296875, 0.859375, 0.171875)^\top$.

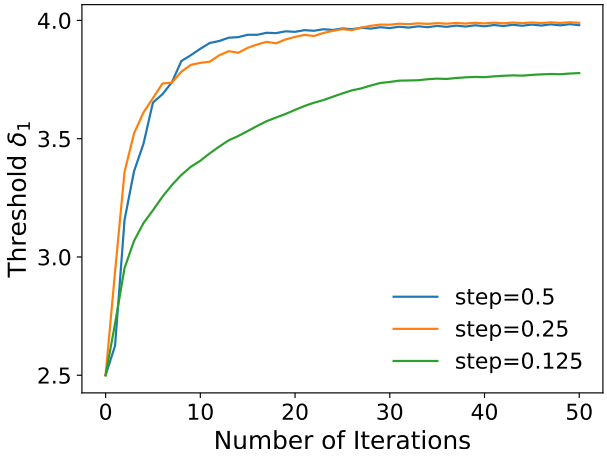


Figure 5.4: In each iteration threshold δ_1 is pushed up.

Notice that, in each iteration, the threshold $\delta_1 = \min\{Y_1^\top X, Y_2^\top X\}$ is pushed up. Figure 5.4 presents how the threshold δ_1 is pushed up in each iteration. And after several iterations, X converges to $(1, 0, 1, 0)^\top$.

5.4 Experiment

In this section, we conduct experiments to evaluate our proposed LP program solution, in the following aspects: (I) sensor selection for bursty cascade detection and (II) computational cost.

5.4.1 Dataset

We conduct experiments on two datasets: a Singapore-based Twitter dataset and a Shanghai-based Weibo dataset. For the Twitter dataset, we crawled Twitter users whose profile locations are Singapore from a seed set of local celebrities and active users. We traced their follower/followee links by two hops. In this way we obtained 184,794 Twitter users. In similar way, we also obtained 105,142 Shanghai-based Weibo users. Then tweets are crawled from these users in a period of five months. In all the Singapore-based Twitter dataset contains 32,479,134 tweets, and the Shanghai-based Weibo dataset contains 19,482,504 tweets. From these tweets, we extracted all the URL links and hashtags. These URL links and hashtags are considered as the identities of cascades. In other words, all the tweets which contain the same URL link (or the same hashtag) represent a cascade.

Table 5.2: Dataset.

Dataset	Type	Training	Testing
Twitter	URL	6,452,732	1,657,145
	hashtag	540,115	190,420
Weibo	URL	1,894,226	650,126
	hashtag	405,411	155,789

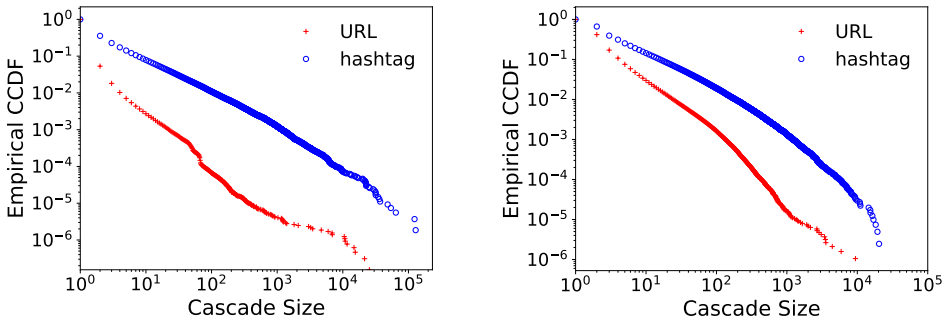


Figure 5.5: Cumulative distribution of the cascade sizes: (a) Twitter (b) Weibo.

For both datasets, we split the data set into two parts: four months data for training and the last one month data for testing. Table 5.2 shows the number

of cascades in the experiment. Figure 5.5 presents the cumulative distribution of the sizes of these cascades. It shows that large cascades are rare, which implies there are actually few bursty cascades among the whole population.

5.4.2 Bursty Cascade Detection

5.4.2.1 Preparation

In the experiment, we conduct two types of evaluations: sensor selection (I) for the classifier with *additive* function and (II) for the classifier with *non-additive* function.

For the case of additive function, we adopt the acceleration defined in [97] as the burstiness evaluating function f. The equation of acceleration $\hat{a}$ is presented in Section 5.2.2. As mentioned in Section 5.2.2, given a cascade c, its acceleration $\hat{a}_c(t)$ is actually a function of time, and it changes over time. Here we calculate $f(c(t)) = \hat{a}_c(t)$ as its burstiness score. (A real example is provided in Figure 5.12.)

For the case of non-additive function, we employ the significance score proposed in [82]. It works as follows. For a cascade c, count the daily frequency of tweets in c. Denote it as $count_t$, where t is its corresponding date. Then find the date when its daily count is far beyond its average. We present it in the following equation.

$$sig_\beta(count_t) = \frac{count_t - max\{\mu, \beta\}}{\sigma + \beta}$$

where $count_t$ is the daily count, μ is the average, σ is the standard deviation and β serves as a noise filter. $sig_\beta(count_t)$ here works like the z-score in the case of normal distribution. For burstiness evaluation, we calculate $f(c(t)) = sig_\beta(count_t)$.

Given a training set of cascades C_{train}, we first get all the bursty cascades $C_{burst} = \{c \in C_{train} | \max_t\{f(c(t))\} \geq \delta_0\}$. For case (I), we let $f =$ acceleration,

and use f = acceleration to identify the bursty cascades. For case (II), we let $f^\star$= acceleration, f = significance score, and use f = significance score to identify the bursty cascades. For each cascade $c \in C_{burst}$, we also get its detection time DT_c. Then we construct the burstiness score vector $Y_c = \{f(c_u(DT_c))\}_{u \in U}$ as the input of Algorithm 4. For both cases, acceleration $\hat{a}_c(t)$ is used as the burstiness evaluation function. For each cascade $c \in C_{burst}$, we decompose it into $\bigcup_{u \in U} c_u$. As $\hat{a}_c(t)$ is additive, $\hat{a}_c(DT_c) = \sum_{u \in U} \hat{a}_{c_u}(DT_c)$. Therefore, we calculate $f(c_u(DT_c)) = \hat{a}_{c_u}(DT_c)$. Finally we can get $Y_c = \{f(c_u(DT_c))\}_{u \in U}$. Obviously, for the user u who does not join the cascade c, i.e. $u \notin c$, $f(c_u(DT_c)) = 0$. Therefore, Y_c is sparse.

As our goal in this dissertation is to find a good way to apply existing detection solutions under the constraint of limited resources, for the given testing cascade set C_{test}, we just need to generate the testing labels. Particularly, given a classifier $F_0 = \langle f, \delta_0 \rangle$, for each cascade $c \in C_{test}$, if $F_0(c) = 1$, i.e. $f(c) = \max_t\{f(c(t))\} \geq \delta_0$, we label it as a positive case, i.e. a bursty cascade; otherwise, it is a negative case. For case (I), we let f = acceleration; for case (II), we let f = significance score. Table 5.3 summarizes the burstiness evaluating functions adopted in the experiment.

Table 5.3: Burstiness evaluating functions adopted in the experiment.

	Training	Testing
Case (I) Additive Function	f = acceleration	f = acceleration
Case (II) Non-additive Function	$f^\star$= acceleration f = significance score	f = significance score

5.4.2.2 Baselines

We consider the following baselines.

- **CELF**: Leskovec et al. study the general problem of detecting outbreaks in networks, and formulate this problem as a objective function optimization problem [64]. In their work, three objective functions are proposed. In this baseline, we consider the detection time as the objective function.

The underlying logic is that, if a cascade can be detected early, it should be identified as a burst early too. Denote the initial time of a cascade c as $t_c = min_{\langle d_u, d_t \rangle \in c}\{d_t\}$. Specifically, we use the objective function of a set of sensors S as follows.

$$\pi(S) = \sum_{c \in C} \frac{1}{1 + min_{d_u \in S}\{d_t - t_c\}}$$

If there is no $d_u \in S$ in c, $min_{d_u \in S}\{d_t - t_c\} = \infty$. It can be proved that $\pi(S)$ is a monotone submodular function. The CELF algorithm is used to find a set of m sensors to maximize $\pi(S)$.

- **CELF***: The cascades with large number of users may be more important than small cascades. In [106] a slight different objective function which involves the size of each cascade is proposed.

$$\pi(S) = \sum_{c \in C} \frac{|c|}{1 + min_{d_u \in S}\{d_t - t_c\}}$$

This $\pi(S)$ is also a monotone submodular function. We apply the same greedy algorithm as above to get a set of m sensors.

- **In / Out Degree**: As mentioned in Section 2.2, a common heuristic strategy is to select the central users within the social network as sensors because information can easily spread to them. When the entire network is not available, a technique inspired by the "friendship paradox" can be applied to sample central users from a network [18]. Since we crawled all the links between the users, in this baseline we simply select top m users according to their in / out degrees.

- **Random**: It is not a bad choice to select users randomly from the whole population. The reason is that, by uniform random sampling, we can get the unbiased estimation of the arrival rate, which is an important indicator of burst. In this baseline we simply randomly select m users.

5.4.2.3 Detection Performance

We run our LP solution as well as all the baselines on training data and each solution selects a set of sensors. Then we test these sensors on the testing data. For a given set of sensors $S \subset U$ and a classifier $F_0 = \langle f, \delta_0 \rangle$, we evaluate its quality as follows. First, for each cascade $c \in C_{test}$, according to burstiness evaluating function f, calculate the burstiness score of the sub-cascade c_S, i.e. $f(c_S)$. Then, based on these burstiness scores $\{f(c_S)\}$ and the testing labels $\{F_0(c)\}$, draw its ROC curve, and calculate the area under the curve (AUC). The larger the AUC is, the better the selected set is.

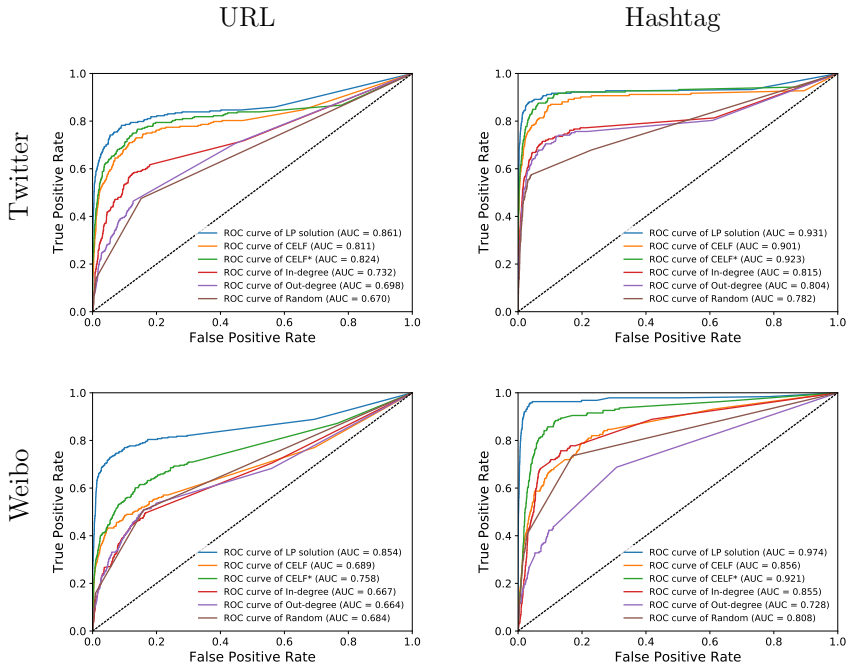


Figure 5.6: The ROC curves of different solutions on different sets of cascades.

For case (I) where f is additive, we have the following results. Figure 5.6 shows the ROC curves of different solutions on different datasets when budget $m = 3000$. We can see that the results are consistent for both URL cascades and hashtag cascades. And our proposed LP solution outperforms all other solutions. The reason is probably that our proposed LP solution is designed to find the best set of sensors for burst detection, while other solutions are relatively intuitive. It is also can be observed that performance of CELF* is

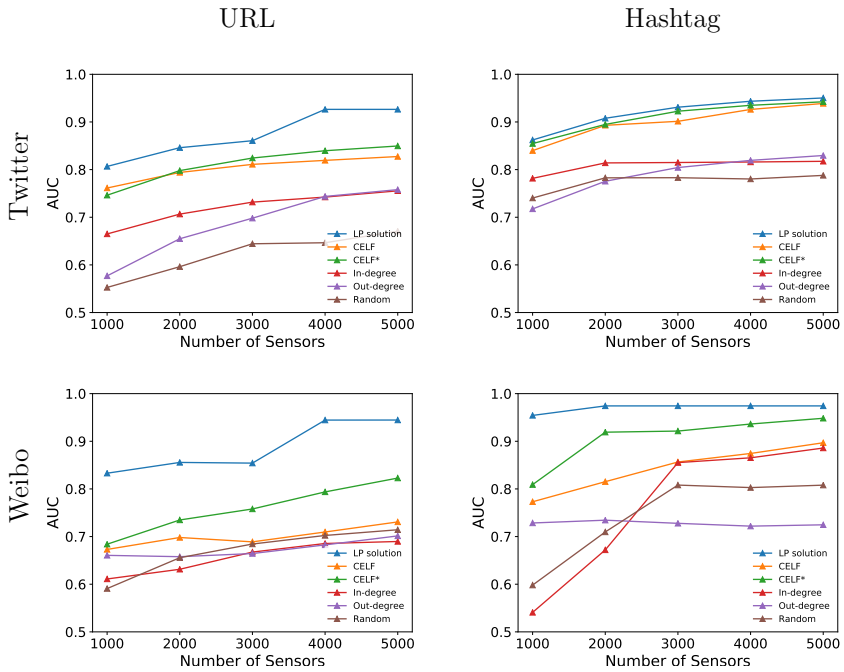


Figure 5.7: Varying budget m, AUC of different solutions on different sets of cascades.

consistently better than the performance of CELF. One possible reason is that the sensors which CELF* locates are more sensitive to large cascades, and nearly all the bursty cascades are large cascades.

We also study the effect of budget m. Figure 5.7 shows the AUC of our proposed LP solution and other baselines for different m. It is can be observed that our proposed LP solution outperforms other baselines consistently when m varies. And as expected, it shows that better performance can be achieved when we set larger budget m.

Particularly, it can be observed that, for the hashtag cascades of both Twitter and Weibo datasets, the performance of our solution is quite good (See the second column in Figure 5.7). Notice that, 5000 is the maximum number of users one can follow in Twitter[2]. It means that, by investing one Twitter account which follows our selected 5000 users, we can detect most of the bursty hashtag cascades in Singapore.

For case (II) where f is non-additive, we have similar results, which are

[2]https://support.twitter.com/articles/68916

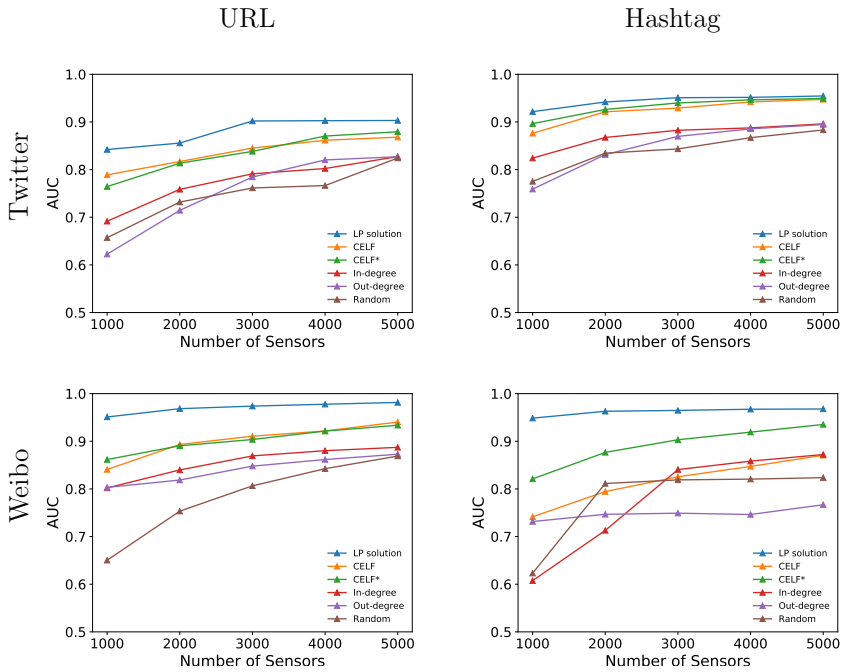


Figure 5.8: Varying budget m, AUC of different solutions on different sets of cascades (for non-additive function).

presented in Figure 5.8. These results support our analysis in Section 5.2.5.

5.4.2.4 Detection Time

Besides the detection performance, detection time is also important because there is no value if the detection delay is too long. In Section 5.2.4, we impose the constraints $DT_{c_S} \leq DT_c$ to maker sure for any cascade $c \in C_{train}$ the detection time of c_S is at least as early as the detection time of c. Here we check whether $DT_{c_S} \leq DT_c$ is still hold on testing cascades C_{test}. Different from other baseline methods, besides the set of users S, LP solution also learns the new threshold δ_1. So it is easy for us to compare the difference of detection time. Particularly, we check each bursty cascade detected by LP solution, i.e. $\{c \in C_{test} | f(c) \geq \delta_0 \wedge f(c_S) \geq \delta_1\}$, and calculate the difference $DT_{c_S} - DT_c$. A negative difference means we can detect the burst on a sub-cascade c_S even earlier than on the original cascade c.

Figure 5.9 presents the difference of detection time, i.e. $DT_{c_S} - DT_c$ for different datasets. It can be observed that the median lines (i.e. yellow lines

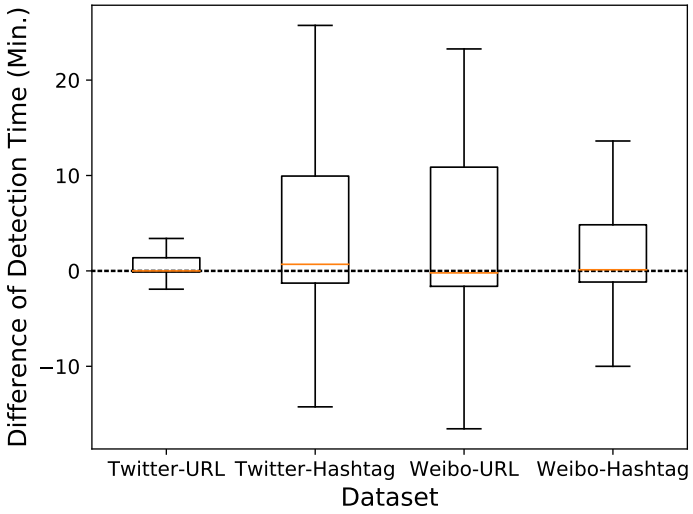


Figure 5.9: The Difference of Detection Time: $DT_{c_S} - DT_c$.

in the box plots) are near to 0, which means for half of the cascades, a burst is detected on a sub-cascade c_S even earlier than it is detected on the original cascade c. And for most bursty cascades, this difference is less than 10 minutes.

5.4.2.5 Empirical Bound

In our LP solution, we take several steps of relaxation: from the original goal — Problem 5.1 to a more practical Problem 5.2 which has a different threshold δ_1, to Problem 5.3 which has less constraints, to linear version Problem 5.4 and 5.5, to Problem 5.6 resulting from linear programming relaxation, finally to Problem 5.7 which involves detection time. One natural question is that how close is our solution to the solution of the original problem, or is there any bound? As mentioned in Section 5.2.1, a feasible solution may not exist in Problem 5.1. So here we discuss how close is our solution, i.e. the solution of Problem 5.7 to the solution of Problem 5.2. Particularly, we empirical check how well our solution approximates the optimal solution. It is hard to directly solve Problem 5.2 in a reasonable time because it is a constraint satisfaction problem which has large number of constraints. So instead of directly solving Problem 5.2, we calculate the AUC of our solution based on training data.

First, we apply Algorithm 4 on C_{train} to choose a set of users S. Then for each cascade $c \in C_{train}$, according to burstiness evaluating function f, calculate the burstiness score of the sub-cascade c_S, i.e. $f(c_S) = \max_t\{f(c_S(t))\}$. Based on these burstiness scores $\{f(c_S)\}$ and the training labels $\{F_0(c)\}$, calculate the AUC. If AUC $= 1$, it means there is a threshold δ_1 splits C_{train} perfectly with true positive rate 1 and false positive rate 0. In other words, if AUC=1, then S is a solution of Problem 5.2.

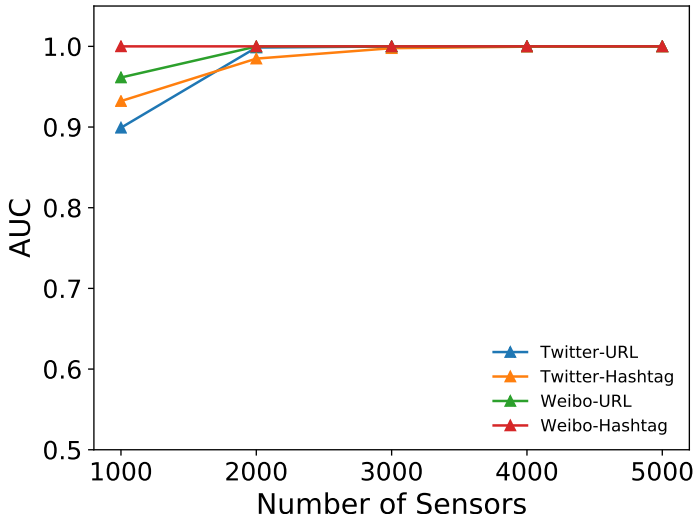


Figure 5.10: Varying budget m, AUC of LP solution on training cascades.

Figure 5.10 presents the AUC of our solutions for different datasets. (Notice that Figure 5.10 is based on training datasets, while Figures 5.7 and 5.8 are based on testing datasets.) For most cases, when $m \geq 2000$, the AUC value is close to 1 (i.e. above 0.9). And for several cases, take Weibo-Hashtag dataset for instance, the AUC value is exactly 1 when $m \geq 3000$. It shows that our solution is close to the optimal one.

5.4.3 Efficiency

We run the experiment on a 64-bit addressing Intel Xeon 3.06 GHz machine. We compare the run time of our proposed sub-gradient method with simplex method, interior-point method as well as CELF. We implement our pro-

posed sub-gradient method in Matlab. For simplex method and interior-point method, we simply call the function *linprog* in Matlab's optimization toolbox. We also implemented CELF in C++. For all above methods, a step of loading data is required. For CELF, we need to scan all the cascades to build the inverted index. For our LP solution, we need to scan all the cascades to identify the bursty cascades and then construct the burstiness score vector Y_c. Here the loading time is not included in the run time. It takes hours for CELF to select 1000 sensors and nearly one day to select 5000 sensors from millions of cascades. As its run time is not at the same scale as other methods, here we just present the run time for other three methods.

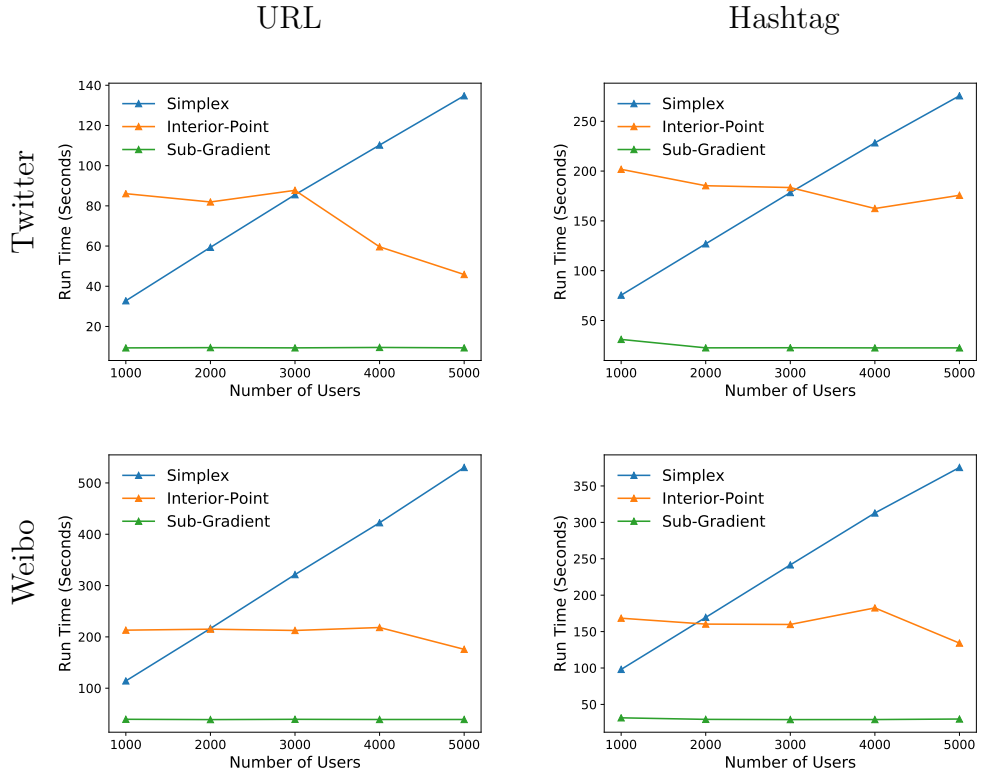


Figure 5.11: Run time of Simplex Method, Interior-Point Method and Sub-Gradient Method.

Figure 5.11 presents the 10-times average run time of simplex method, interior-point method and sub-gradient method on different datasets. We can see that our proposed sub-gradient method is more efficient than simplex method and interior-point method. Moreover, the run time of sub-gradient

Table 5.4: Jaccard coefficients of the selected users from different methods.

	CELF	CELF*	In-Deg	Out-Deg	Random
LP	0.1062	0.1050	0.0241	0.0246	0.0055
CELF	-	0.1947	0.0368	0.0352	0.0060
CELF*	-	-	0.0390	0.0368	0.0055
In-Deg	-	-	-	0.3587	0.0055
Out-Deg	-	-	-	-	0.0101

method is steady when m varies. In contrast, the run time of other two methods changes significantly for different m.

5.4.4 Comparison of the Selected Users

In this subsection, we examine how the methods select users differently. Table 5.4 presents the Jaccard coefficients of the selected 1000 Twitter users from different methods on Twitter-Hashtag cascades. It shows that different methods select users quite differently. For example, comparing LP with other methods, the largest Jaccard coefficient value (between CELF and LP) is 0.1062, which is pretty low. Furthermore, we study the following characteristics of these selected users:

- User activities: (I) how many tweets a user generates and (II) the retweet ratio, i.e. the ratio of the number of retweets of a user to the total number of tweets of a user.
- Immediate network: (I) the in-degree (i.e. number of friends), and (II) the out-degree (i.e. number of followers).
- Account type: we manually check whether the account is a verified news media or not.

Table 5.5 presents the statistics of the selected users from different methods. We can see that the users selected by LP generate more tweets and have higher retweet ratio than the users selected by other methods. It is interesting because we do not explicitly model the activeness of users in LP. It also can be observed

Table 5.5: Statistics of the selected users from different methods.

	LP	CELF	CELF*	In-Deg	Out-Deg	Random
Median of number of tweets	4747.0	4549.5	3420.5	472.5	486.0	177.5
Median of retweet ratio	0.2526	0.1863	0.1887	0.1041	0.0774	0.0903
Median of number of friends	282.0	326.0	297.5	1044.0	622.0	149.0
Median of number of followers	256.0	333.0	311.0	778.5	1607.0	126.0
Number of news media	6	1	5	2	18	0

that LP selects more news media accounts than other methods, except for Out-Degree. These selected accounts include "ChannelNewsAsia", "STcom", "TODAYonline", which are the most influential news media in Singapore.

For the purpose of demonstration, we pick a cascade of *#hougangbyelection*, which is the most popular bursty hashtags in our data set (in terms of the number of relevant tweets). This hashtag is about a local by-election in Singapore. Around 22:30 at that day the election result came out, and it triggered a surge of relevant tweets within a short period of time. Figure 5.12 presents the cascade size, the arrival rate (i.e. the number of relevant tweets per minute) and the acceleration (with smoothing window $\Delta T_1 = 30$ mins, $\Delta T_2 = 60$ mins) of this cascade, as well as the sub-cascades observed from the users selected by different methods. Both arrival rate and acceleration reflect the burstiness of a cascade. Although different methods select the same number of users, we can see that, for both cases, the peaks of LP in Figure 5.12 (second row) are higher than others' to a large extent.

5.5 Limitation

It is a common way to look at the past to predict the future. In this work, we follow this way, i.e. choose a budgeted set of users according to previous data (or training data) with the hope that we can detect most bursty cascades from these users on future data (or test data). If the characteristic of data is consistent over time, it will not be a problem. However, if the characteristic of data suddenly changes, there will be a gap between the training data and test

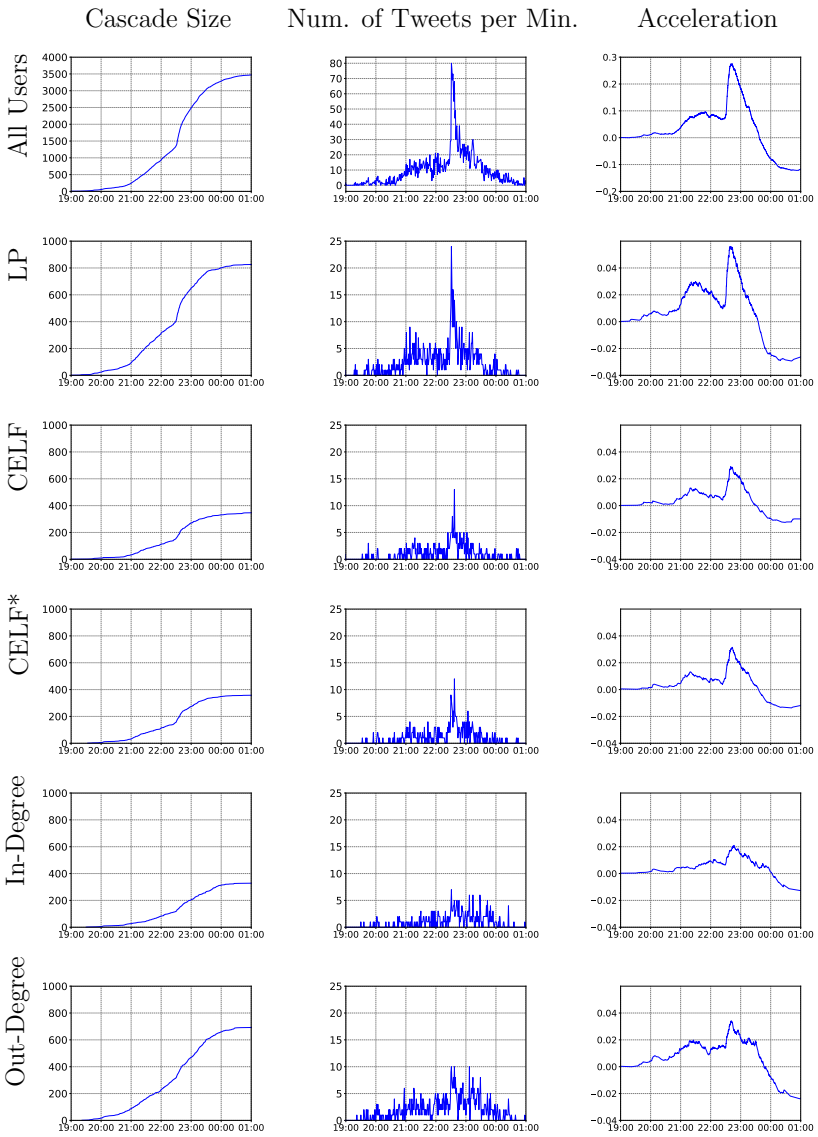


Figure 5.12: The cascade size, arrival rate and acceleration of the sub-cascades from users selected by different methods.

data, which may make our algorithm fail. For example, a user may close her account for some reason after actively sharing information in microblogs. In this case, our algorithm may fail because it would choose this user as a social sensor for future social monitoring. In order to cope with this limitation, it is suggested to select users regularly based on new data, which reflects the change of the data characteristic.

Besides, another limitation of our current work is that only two burstiness evaluating functions are explored. We plan to test more burstiness evaluating functions in the future.

5.6 Summary

In this chapter, we propose a general sensor selection problem for different burst detection approaches. In general, sensor selection problem is NP-hard. Especially for the large social networks with millions of users, existing greedy methods are hardly scale to such size. After formulating this problem as a constraint satisfaction problem, we transform it into an LP (Linear Programming) problem which has only few constraints. Furthermore, we develop a sub-gradient algorithm to solve the LP problem, which makes it possible for our solution to scale up to large social networks. Compared with existing solutions, our solution can find better set of sensors for burst detection.

Chapter 6

Conclusion

6.1 Dissertation Summary

In the following paragraphs, we would like to summarize milestone accomplishments in this dissertation. Our research is motivated by the emergence of microblogging services such as Twitter and Sina Weibo. Beyond traditional channels such as newspaper and television, these social platforms provide us more timely information sources. We start our study with the following research questions:

1. Can we leverage microblogging services for automated bursty topic detection in real-time?

2. How to select a budgeted set of social network sensors to form the data stream for bursty cascade detection without compromising the detection performance?

3. How many users will a cascade eventually reach in microblogs? Or will it go viral?

Chapter 3 provides an answer for the first question. We propose TopicSketch a framework for real-time detection of bursty topics from Twitter. Due to the huge volume of tweet stream, existing topic models can hardly scale to data of

such sizes for real-time topic modeling tasks. We develop a "sketch of topic", which provides a "snapshot" of the current tweet stream and can be updated efficiently. Once burst detection is triggered, bursty topics can be inferred from the sketch efficiently. Compared with existing event detection system, from a different perspective — the "accelerations of topics", our solution can detect bursty topics in real-time, and present them in finer-granularity.

To answer the second question, in Chapter 5 we propose a general sensor selection problem for different burst detection approaches. In general, sensor selection problem is NP-hard. Especially for the large social networks with millions of users, existing greedy methods are hardly scale to such size. After formulating this problem as a constraint satisfaction problem, we transform it into an LP (Linear Programming) problem which has only few constraints. Furthermore, we develop a sub-gradient algorithm to solve the LP problem, which makes it possible for our solution to scale up to large social networks. Compared with existing solutions, our solution can find better set of sensors for burst detection.

We provide an answer for the last question in Chapter 4 by using a general time-aware cascade model to describe the dynamic process of growing cascades in social networks over time. Based on this general model, a concrete model TCM-LH is designed for the retweeting cascades in Twitter. Our experiment results show our proposed TCM-LH fits the real cascade data better than other baselines in terms of model fitting. More importantly, we empirically show that our proposed TCM-LH could help to improve the performance of virality prediction.

6.2 Future Work

We conclude this dissertation by outlining several promising research directions for further exploration.

In this dissertation, we leverage microblogging services for early detection of bursty topics from a pure *on-line* perspective. Particularly, we detect bursty topics from tweets generated by all the on-line accounts in cyberspace. When we were concluding this dissertation, we realized that, what behind all these on-line accounts are millions or even billions of *off-line* live users in the real world, which drive all the real-life business applications. This means that much more value could be brought out if we "embrace" the off-line users in the real world with the on-line discovery in cyberspace. Our previous work [95] has taken the first step — identifying a users off-line real-life social community from examining its on-line social network structure. Such works which bridge the on-line cyberspace and off-line real world are important. For example, a common way to measure the impact of a certain bursty topic is counting the total number of relevant tweets, which can be easily abused. Because off-line social ties are stronger than on-line ones, with the knowledge of off-line social network, we can design a more sophisticated way to evaluate the impact of this bursty topic by considering users' off-line neighborhood.

Besides, we would also like to build a user preference-aware model to enhance our current time-aware model for better prediction performance. In our current model, for each particular cascade, every user in the social network share the same hazard function. However, in real life, different users have different behavior patterns – some users may be more active while other users may be more likely to consume information rather than to share information. More importantly, different users have different preferences. For example, driven by their own interests, some users may be more likely to share information about football while other users may share more tweets about fashion. Therefore, we expect that, in consideration of user preference, the prediction performance of our current model can be potentially improved.

Appendix A

Acceleration Calculation

In Chapter 3, we played a "*acceleration*" trick to preserve the information of bursty topics but filter out the others. In this section, we briefly introduce the concept of "*acceleration*", and the related concept "*velocity*" in the context of stream data, and more important, how we estimate them in a framework of convolutional smoothing.

Consider a sequence of items $\{X_i\}_{i\in\mathcal{I}}$, where $\mathcal{I}$ is the set of index. Each item X_i can be arbitrary non-negative real value, e.g. the frequency of a word in a tweet. t_i is the corresponding timestamp. Note here the timestamp t_i is continuous, rather than discrete time unit. For simplicity's sake, we use $\{X_i\}$ instead of $\{X_i\}_{i\in\mathcal{I}}$.

We first define the accumulated *volume* function $s(t)$ as

$$s(t) = \sum_{t_i \le t} X_i,$$

which is the accumulation of X_i by time t. Following the mathematical definitions of "*velocity*" $v(t)$ and "*acceleration*" $a(t)$, which are the first order derivative and second order derivative of $s(t)$ respectively, we have $v(t) = s'(t)$, $a(t) = s''(t) = v'(t)$.

As $s(t)$ is a step function, directly taking derivative of $s(t)$ just gets several spikes along the timeline. We consider the following convolution as a smoothing

operation on $s(t)$.

$$(s * g_1)(t) = \int_0^t s(\tau) g_1(t-\tau) \mathrm{d}\tau$$

where $*$ is convolution operation, and g_1 is a smoothing function, which satisfies $\int_0^\infty g_1(\tau)\mathrm{d}\tau = 1$. We estimate v as

$$\hat{v} = (s * g_1)' = s' * g_1 \tag{A.1}$$

In the same way, to smooth $\hat{v}(t)$, we take another convolution operation on $\hat{v}(t)$ as following.

$$(\hat{v} * g_2)(t) = \int_0^t \hat{v}(\tau) g_2(t-\tau) \mathrm{d}\tau$$

where g_2 is another smoothing function. We estimate a as

$$\begin{aligned} \hat{a} &= (\hat{v} * g_2)' = ((s' * g_1) * g_2)' = (s' * (g_1 * g_2))' \\ &= s' * (g_1 * g_2)' \end{aligned} \tag{A.2}$$

As $s(t)$ is a step function,

$$s'(t) = \sum_{t_i \le t} X_i \cdot \delta(t - t_i)$$

where $\delta()$ is the Dirac delta function. As $\delta * g = g$, we estimate the velocity $v(t)$ and the acceleration $a(t)$ as follows.

$$\hat{v}(t) = s' * g_1 = \sum_{t_i \le t} X_i \cdot g_1(t - t_i) \tag{A.3}$$

$$\hat{a}(t) = s' * (g_1 * g_2)' = s' * g_0 = \sum_{t_i \le t} X_i \cdot g_0(t - t_i) \tag{A.4}$$

where $g_0 = (g_1 * g_2)'$.

In this work, as shown in Equation A.4, we treat the **acceleration** on an item stream $\{X_i\}$ at time t **as a weighted sum of** $\{X_i\}$, where the function $g_0 = (g_1 * g_2)'$ defines the weights. In general, we denote $\mathcal{A}_t(\{X_i\}) = \hat{a}(t)$ in

Equation A.4 as a linear operation on stream $\{X_i\}$ based on some smoothing function.

For instance, in this framework, when set $g_1(t) = I(0 \leq t < \Delta T)/\Delta T$, where $I(\cdot)$ is the indicator function, according to Equation A.3, we get the continuous version of Moving Average (MA) as the velocity. When set $g_1(t) = exp(-t/\Delta T_1)/\Delta T_1$, $g_2(t) = exp(-t/\Delta T_2)/\Delta T_2$ (where $\Delta T_1 \neq \Delta T_2$), from Equation A.3 and A.4, we can derive Equation 3.1 in Section 3.2.1.

Appendix B

Infer Topics from Sktech

The tool we use to infer topics from sketch is tensor decomposition [5]. Consider we have the following sketch M_2 and M_3.

$$M_2 = \sum_{k=1}^{K} a_k \cdot \phi_k \otimes \phi_k = \sum_{k=1}^{K} a_k \cdot \phi_k \phi_k^\top$$

$$M_3 = \sum_{k=1}^{K} a_k \cdot \phi_k \otimes \phi_k \otimes \phi_k$$

where $a_k \in \mathbb{R}$, $\phi_k \in \mathbb{R}^N$, $\sum_{w=1}^{N} \phi_{k,w} = 1$. And assume that $a_k \neq 0$ and $\{\phi_k\}_{k=1}^K$ are linear independent. Note that here a_k can be negative, which is weaker than the non-degeneracy assumption in [5]. Project M_3 to a matrix as follows:

$$M_3(\eta) = \sum_{k} a_k \cdot \phi_k \phi_k^\top \langle \eta, \phi_k \rangle$$

where $\eta \in \mathbb{R}^N$, is a random vector.

First $rank(M_2) = K$. Short proof here. For any $x \in \mathbb{R}^N$, if $M_2 x = \sum_{k=1}^{K} a_k \cdot \langle x, \phi_k \rangle \phi_k = 0$, as $\{\phi_k\}_{k=1}^K$ are linear independent, $a_k \cdot \langle x, \phi_k \rangle = 0$, $a_k \neq 0$, so for all the k, $\langle x, \phi_k \rangle = 0$. It means the null space of M_2, $Null(M_2) = span(\{\phi_k\}_{k=1}^K)^\perp$, where $span(\{\phi_k\}_{k=1}^K)^\perp$ is the orthogonal complement of the $span(\{\phi_k\}_{k=1}^K)$. So the dimension of $Null(M_2)$, $nullity(M_2) = N - K$. So $rank(M_2) = N - nullity(M_2) = K$.

As M_2 is a real symmetric matrix, and $rank(M_2) = K$, so that M_2 has k nonzero real eigenvalues $\{\lambda_k\}_{k=1}^K$, and $M_2 = \sum_{k=1}^K \lambda_k \cdot u_k u_k^\top$, where $\{u_k\}_{k=1}^K$ are corresponding eigenvectors, and they are orthonormal. It is easy to show that $Null(M_2) = span(\{u_k\}_{k=1}^K)^\perp$, which means $span(\{\phi_k\}_{k=1}^K) = span(\{u_k\}_{k=1}^K)$.

Whitening

Let $W = \left(u_1/\sqrt{\lambda_1}, ..., u_K/\sqrt{\lambda_K}\right)$, we have

$$W^\top M_2 W = I_{K\times K}$$

Note that W is a complex matrix, because λ_k may be a negative value. So

$$W^\top M_2 W = \sum_{k=1}^K a_k \cdot (W^\top \phi_k)(W^\top \phi_k)^\top = I_{K\times K}$$

It implies for any $k_1 \neq k_2$, $(W^\top \phi_{k_1})^\top (W^\top \phi_{k_2}) = 0$, and $(W^\top \phi_k)^\top (W^\top \phi_k) = a_k^{-1}$.

Whiten $M_3(\eta)$ as follows.

$$T_3 = W^\top M_3(\eta) W = \sum_k a_k \cdot \langle \eta, \phi_k \rangle (W^\top \phi_k)(W^\top \phi_k)^\top$$

Tensor Power Method

$$\begin{aligned} T_3(W^\top \phi_k) &= \sum_k a_k \cdot \langle \eta, \phi_k \rangle (W^\top \phi_k)(W^\top \phi_k)^\top (W^\top \phi_k) \\ &= \langle \eta, \phi_k \rangle (W^\top \phi_k) \end{aligned}$$

Under the strict condition of distinction, i.e. for any $k_1 \neq k_2$, $\langle \eta, \phi_{k_1} \rangle \neq \langle \eta, \phi_{k_2} \rangle$, T_3 has different eigenvalues $\{\langle \eta, \phi_k \rangle\}_{k=1}^K$ and corresponding eigenvectors $\{W^\top \phi_k\}_{k=1}^K$. Denote the generalized eigenvectors of T_3 as $\{v_k\}_{k=1}^K$, so $v_k = c_k W^\top \phi_k$, where c_k is some complex number.

Reconstruction

As $span(\{\phi_k\}_{k=1}^K) = span(\{u_k\}_{k=1}^K)$, $\phi_k \in span(\{u_k\}_{k=1}^K)$, it means exists some $b_k \in \mathbb{R}^K$ such that $\phi_k = Wb_k$. So $v_k = c_k W^\top \phi_k = c_k W^\top W b_k$, $c_k b_k = (W^\top W)^{-1} v_k$, $c_k \phi_k = c_k W b_k = W(W^\top W)^{-1} v_k$. Denote $\tilde{v_k} = W(W^\top W)^{-1} v_k$. As $\sum_{w=1}^N \phi_{k,w} = 1$, $c_k = \sum_{w=1}^N \tilde{v_k}$, i.e. $c_k = 1_N^\top W(W^\top W)^{-1} v_k$. At last, we have

$$\phi_k = \frac{W(W^\top W)^{-1} v_k}{1_N^\top W(W^\top W)^{-1} v_k}$$

$$a_k = \frac{1}{(W^\top \phi_k)^\top (W^\top \phi_k)}$$

Appendix C

Submodular Function

For the sensor selection problem (or sensor placement problem) in networks, greedy algorithm is a common solution. As long as a monotone submodular objective function can be constructed, the error bound $1-1/e$ is guaranteed [81]. However, it is not easy to construct such a monotone submodular objective function for burst detection.

Here we consider a very simple classifier $F_0 = \langle f, \delta_0 \rangle$, where $f = |c|$ and $\delta_0 = 2$. It means, if a cascade is observed twice, it will be detected as a burst. Similar to [64], we can construct the following objective function for a cascade c.

$$\pi(S) = \frac{1}{1 + t_{S,c}} \tag{C.1}$$

where $t_{S,c}$ is burst detection time, i.e. the second minimum of $\{t_u\}_{u \in S \wedge \langle u, t_u \rangle \in c}$.

According to [81], for a finite set S, a submodular function is a set function $\pi : 2^U \to \mathbb{R}$, which satisfies the following definition: *for every* $S \subseteq U$ *and* $v_1, v_2 \in U \setminus S$ *we have that* $\pi(S \cup \{v_1\}) + \pi(S \cup \{v_2\}) \geq \pi(S \cup \{v_1, v_2\}) + \pi(S)$.

Here we show that the above $\pi(S)$ in Equation C.1 is not a submodular function by providing a counterexample. Assume u_1, u_2, v_1, v_2 join c at $t_{u_1}, t_{u_2}, t_{v_1}, t_{v_2}$ respectively, and $u_1, u_2 \in S$, $v_1, v_2 \notin S$. Without loss of generality, assume $t_{u_1} = t_{u_2} <$ others, which means detection time $t_{S,c} = t_{u_1} = t_{u_2}$.

Suppose $t_{v_1} < t_{v_2} < t_{u_1} = t_{u_2}$. So $\pi(S) = \frac{1}{1+t_{u_2}}$,$\pi(S \cup \{v_1\}) = \frac{1}{1+t_{u_1}}$, $\pi(S \cup \{v_2\}) = \frac{1}{1+t_{u_1}}$, $\pi(S \cup \{v_1, v_2\}) = \frac{1}{1+t_{v_2}}$. Therefore, we have $\pi(S \cup \{v_1\}) + \pi(S \cup \{v_2\}) < \pi(S \cup \{v_1, v_2\}) + \pi(S)$, which means $\pi(S)$ is not a submodular function.

Appendix D

Hazard Function

In Chapter 4 we use hazard function in survival analysis [68] to model time in cascade. Here we first briefly introduce the basic concepts of survival analysis, then give the formal definition of hazard function.

Survival analysis focuses on time-to-event data, especially the survival time until an event of failure. While typical examples of events of interest are biological death and the failure in mechanical systems [49], survival analysis is in fact generic and can be applied to model any time-to-event data.

Consider as a random variable T the time when an event of interest (e.g., when a Twitter user retweets one particular tweet) happens . The probability that the event happens before a certain time t is $P(T \leq t) = F(t)$, which is the cumulative distribution function (CDF). Without loss of generality, suppose $F(0) = 0$. The probability density function (PDF) $f(t)$ is defined as the derivate of $F(t)$, i.e. $f(t) = \frac{\mathrm{d}F(t)}{\mathrm{d}t}$.

When processing time-to-event data, given that the event has not happened by time t, the probability that this event happens in the next time slice $(t, t+dt]$ is usually of great interest, i.e., $P(t < T \leq t + dt | T > t)$. For example, when we observe a Twitter user has not retweeted one particular tweet by time t, it is highly critical to estimate the probability this Twitter user retweets it in the next time slice $(t, t + dt)$. The **hazard function** (or **hazard rate**) [68] is

defined as

$$h(t) = \lim_{dt \to 0} \frac{P(t < T \leq t + dt | T > t)}{dt} = \frac{f(t)}{1 - F(t)}.$$

The hazard function $h(t)$ reflects the chance that the event happens immediately after time t. By integrating $h(t)$, we have the cumulative hazard function $H(t) = \int_0^t h(u)\mathrm{d}u = \int_0^t \frac{F'(u)}{1-F(u)}\mathrm{d}u = -log(1 - F(u))|_0^t = -log(1 - F(t))$. It reveals the essential relationship between the cumulative distribution function $F(t)$ and the cumulative hazard function $H(t)$ in the following equation

$$F(t) = 1 - e^{-H(t)}.$$

In practice, it is usually impossible to know the exact formula of CDF $F(t)$. However, by studying the hazard rate, we can design the formula of hazard function $h(t)$ or cumulative hazard function $H(t)$, and further derive the formula of $F(t)$ to approximate the real distribution. One example is simply to set $H(t)$ as a linear term, i.e. $H(t) = \frac{t}{\lambda}$. Its corresponding CDF is $F(t) = 1 - e^{-\frac{t}{\lambda}}$, which is in fact the exponential distribution with mean λ. Another slightly complicated example is $H(t) = (\frac{t}{\alpha})^\beta$. Its corresponding CDF is $F(t) = 1 - e^{-(\frac{t}{\alpha})^\beta}$, which is simply the Weibull distribution [92, 72] with scale parameter α and shape parameter β. It turns out that the different choices of hazard functions lead to different probability models. In this work, our job is to construct the proper hazard function to approximate the real cascade development in social network.

Bibliography

[1] Amr Ahmed, Qirong Ho, Choon Hui Teo, Jacob Eisenstein, Alexander J. Smola, and Eric P. Xing. Online inference for the infinite topic-cluster model: Storylines from streaming text. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 101–109, 2011.

[2] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 37–45, 1998.

[3] Foteini Alvanaki, Sebastian Michel, Krithi Ramamritham, and Gerhard Weikum. See what's enblogue: real-time emergent topic identification in social media. In *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 336–347, 2012.

[4] Rodrigo A S Alves, Renato Martins Assunção, and Pedro Olmo Stancioli Vaz de Melo. Burstiness scale: A parsimonious model for characterizing random series of events. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1405–1414, 2016.

[5] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M. Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

[6] Peng Bao, Huawei Shen, Junming Huang, and Xueqi Cheng. Popularity prediction in microblogging network: a case study on sina weibo. In *WWW (Companion Volume)*, pages 177–178, 2013.

[7] Albert-Laszlo Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.

[8] David M. Blei and John D. Lafferty. Dynamic topic models. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 113–120, 2006.

[9] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[10] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[11] Thorsten Brants and Francine Chen. A system for new event detection. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28 - August 1, 2003, Toronto, Canada*, pages 330–337, 2003.

[12] Kevin Robert Canini, Lei Shi, and Thomas L. Griffiths. Online inference of topics with latent dirichlet allocation. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, pages 65–72, 2009.

[13] Mario Cataldi, Luigi Di Caro, and Claudio Schifanella. Personalized emerging topic detection based on a term aging model. *ACM TIST*, 5(1):7, 2013.

[14] Jonathan Chang, Jordan L. Boyd-Graber, Sean Gerrish, Chong Wang, and David M. Blei. Reading tea leaves: How humans interpret topic models. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 288–296, 2009.

[15] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 1029–1038, 2010.

[16] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 199–208, 2009.

[17] Justin Cheng, Lada A. Adamic, P. Alex Dow, Jon M. Kleinberg, and Jure Leskovec. Can cascades be predicted? In *WWW*, pages 925–936, 2014.

[18] Nicholas A Christakis and James H Fowler. Social network sensors for early detection of contagious outbreaks. *PloS one*, 5(9):e12948, 2010.

[19] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[20] Peng Cui, Shifei Jin, Linyun Yu, Fei Wang, Wenwu Zhu, and Shiqiang Yang. Cascading outbreak prediction in networks: a data-driven approach. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 901–909, 2013.

[21] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[22] Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[23] Qiming Diao, Jing Jiang, Feida Zhu, and Ee-Peng Lim. Finding bursty topics from microblogs. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers*, pages 536–544, 2012.

[24] Pedro M. Domingos and Matthew Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 57–66, 2001.

[25] P. Alex Dow, Lada A. Adamic, and Adrien Friggeri. The anatomy of large facebook cascades. In *Proceedings of the Seventh International Conference on Weblogs and Social Media, ICWSM 2013, Cambridge, Massachusetts, USA, July 8-11, 2013.*, 2013.

[26] Nan Du, Mehrdad Farajtabar, Amr Ahmed, Alexander J. Smola, and Le Song. Dirichlet-hawkes processes with applications to clustering continuous-time document streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 219–228, 2015.

[27] Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3147–3155, 2013.

[28] Nan Du, Le Song, Hyenkyun Woo, and Hongyuan Zha. Uncover topic-sensitive information diffusion networks. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, pages 229–237, 2013.

[29] Scott L Feld. Why your friends have more friends than you do. *American Journal of Sociology*, pages 1464–1477, 1991.

[30] Wei Feng, Chao Zhang, Wei Zhang, Jiawei Han, Jianyong Wang, Charu Aggarwal, and Jianbin Huang. STREAMCUBE: hierarchical spatio-temporal hashtag clustering for event exploration over the twitter stream. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1561–1572, 2015.

[31] Adrien Friggeri, Lada A. Adamic, Dean Eckles, and Justin Cheng. Rumor cascades. In *Proceedings of the Eighth International Conference on Weblogs and Social Media, ICWSM 2014, Ann Arbor, Michigan, USA, June 1-4, 2014.*, 2014.

[32] Wojciech Galuba, Karl Aberer, Dipanjan Chakraborty, Zoran Despotovic, and Wolfgang Kellerer. Outtweeting the twitterers - predicting information cascades in microblogs. In *WOSN*, 2010.

[33] Manuel García-Herranz, Esteban Moro Egido, Manuel Cebrián, Nicholas A. Christakis, and James H. Fowler. Using friends as sensors to detect global-scale contagious outbreaks. *CoRR*, abs/1211.6512, 2012.

[34] Sharad Goel, Duncan J. Watts, and Daniel G. Goldstein. The structure of online diffusion networks. In *ACM Conference on Electronic Commerce, EC '12, Valencia, Spain, June 4-8, 2012*, pages 623–638, 2012.

[35] André Gohr, Alexander Hinneburg, Rene Schult, and Myra Spiliopoulou. Topic evolution in a stream of documents. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*, pages 859–870, 2009.

[36] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.

[37] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 561–568, 2011.

[38] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 1019–1028, 2010.

[39] Manuel Gomez-Rodriguez, Jure Leskovec, and Bernhard Schölkopf. Modeling information propagation with survival theory. In *ICML (3)*, pages 666–674, 2013.

[40] Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420, 1978.

[41] T.L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(Suppl 1):5228–5235, 2004.

[42] Peter Guttorp. An introduction to the theory of point processes (D. j. daley and d. vere-jones). *SIAM Review*, 32(1):175–176, 1990.

[43] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[44] Dan He and Douglas Stott Parker Jr. Topic dynamics: an alternative model of bursts in streams of topics. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 443–452, 2010.

[45] Matthew D. Hoffman, David M. Blei, and Francis R. Bach. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.*, pages 856–864, 2010.

[46] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, 1999.

[47] Liangjie Hong, Amr Ahmed, Siva Gurumurthy, Alexander J. Smola, and Kostas Tsioutsiouliklis. Discovering geographical topics in the twitter stream. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 769–778, 2012.

[48] Liangjie Hong, Ovidiu Dan, and Brian D. Davison. Predicting popular messages in twitter. In *WWW (Companion Volume)*, pages 57–58, 2011.

[49] David W Hosmer Jr, Stanley Lemeshow, and Susanne May. *Applied survival analysis: regression modeling of time to event data*, volume 618. Wiley. com, 2011.

[50] Alexander Ihler, Jon Hutchins, and Padhraic Smyth. Adaptive event detection with time-varying poisson processes. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 207–216, 2006.

[51] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613, 1998.

[52] Matthew O. Jackson and Leeat Yariv. Diffusion on social networks. *Economie Publique*, 16:3–16, 2005.

[53] Maximilian Jenders, Gjergji Kasneci, and Felix Naumann. Analyzing and predicting viral tweets. In *WWW (Companion Volume)*, pages 657–664, 2013.

[54] C. Jin, W. Qian, C. Sha, J.X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 287–294, 2003.

[55] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[56] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

[57] J. Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[58] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM TIST*, 2(4):32, 2011.

[59] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. On the bursty evolution of blogspace. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, pages 568–576, 2003.

[60] Andrey Kupavskii, Liudmila Ostroumova, Alexey Umnov, Svyatoslav Usachev, Pavel Serdyukov, Gleb Gusev, and Andrey Kustarev. Prediction of retweet cascade size over time. In *CIKM*, pages 2335–2338, 2012.

[61] Wonyeol Lee, Jinha Kim, and Hwanjo Yu. CT-IC: Continuously activated and time-restricted independent cascade model for viral marketing. In *ICDM*, pages 960–965, 2012.

[62] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1):5, 2007.

[63] Jure Leskovec, Lars Backstrom, and Jon M. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 497–506, 2009.

[64] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 420–429, 2007.

[65] Chenliang Li, Aixin Sun, and Anwitaman Datta. Twevent: segment-based event detection from tweets. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 155–164, 2012.

[66] Duc Minh Luu, Ee-Peng Lim, Tuan-Anh Hoang, and Freddy Chong Tat Chua. Modeling diffusion in social networks using network properties. In *Proceedings of the Sixth International Conference on Weblogs and Social Media, Dublin, Ireland, June 4-7, 2012*, 2012.

[67] Michael Mathioudakis and Nick Koudas. Twittermonitor: trend detection over the twitter stream. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 1155–1158, 2010.

[68] Rupert G Miller Jr. *Survival analysis*, volume 66. John Wiley & Sons, 2011.

[69] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243. Springer, 1978.

[70] Seth A. Myers, Chenguang Zhu, and Jure Leskovec. Information diffusion and external influence in networks. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 33–41, 2012.

[71] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, pages 181–189, 2010.

[72] John E Pinder, James G Wiener, and Michael H Smith. The weibull distribution: a new method of summarizing survivorship data. *Ecology*, 59(1):175–179, 1978.

[73] Manolis Platakis, Dimitrios Kotsakos, and Dimitrios Gunopulos. Searching for events in the blogosphere. In *Proceedings of the 18th International*

Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009, pages 1225–1226, 2009.

[74] Matthew Richardson and Pedro M. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 61–70, 2002.

[75] R Tyrrell Rockafellar. Convex analysis, volume 28 of princeton mathematics series, 1970.

[76] Everett M. Rogers. *Diffusion of innovations (5. ed.)*. Free Press, 2003.

[77] Daniel M. Romero, Brendan Meeder, and Jon M. Kleinberg. Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on twitter. In *WWW*, pages 695–704, 2011.

[78] Ankan Saha and Vikas Sindhwani. Learning evolving and emerging topics in social media: a dynamic nmf approach with temporal regularization. In *Proceedings of the Fifth International Conference on Web Search and Web Data Mining, WSDM 2012, Seattle, WA, USA, February 8-12, 2012*, pages 693–702, 2012.

[79] Kazumi Saito, Masahiro Kimura, Kouzou Ohara, and Hiroshi Motoda. Learning continuous-time information diffusion model for social behavioral data analysis. In *ACML*, pages 322–337, 2009.

[80] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 851–860, 2010.

[81] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer Science & Business Media, 2003.

[82] Erich Schubert, Michael Weiler, and Hans-Peter Kriegel. Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 871–880, 2014.

[83] Huijuan Shao, K. S. M. Tozammel Hossain, Hao Wu, Maleq Khan, Anil Vullikanti, B. Aditya Prakash, Madhav V. Marathe, and Naren Ramakrishnan. Forecasting the flu: Designing social network sensors for epidemics. *CoRR*, abs/1602.06866, 2016.

[84] Kyoungwon Suh, Yang Guo, James F. Kurose, and Donald F. Towsley. Locating network monitors: Complexity, heuristics, and coverage. *Computer Communications*, 29(10):1564–1577, 2006.

[85] Lijun Sun, Kay W. Axhausen, Der-Horng Lee, and Manuel Cebrián. Efficient detection of contagious outbreaks in massive metropolitan encounter networks. *CoRR*, abs/1401.2815, 2014.

[86] Gábor Szabó and Bernardo A. Huberman. Predicting the popularity of online content. *CoRR*, abs/0811.0405, 2008.

[87] Yusuke Takahashi, Takehito Utsuro, Masaharu Yoshioka, Noriko Kando, Tomohiro Fukuhara, Hiroshi Nakagawa, and Yoji Kiyota. Applying a burst model to detect bursty topics in a topic model. In *Advances in Natural Language Processing - 8th International Conference on NLP, JapTAL 2012, Kanazawa, Japan, October 22-24, 2012. Proceedings*, pages 239–249, 2012.

[88] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[89] Chong Wang, David M. Blei, and David Heckerman. Continuous time dynamic topic models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, pages 579–586, 2008.

[90] Xuanhui Wang, ChengXiang Zhai, Xiao Hu, and Richard Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 784–793, 2007.

[91] Xuerui Wang and Andrew McCallum. Topics over time: a non-markov continuous-time model of topical trends. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 424–433, 2006.

[92] Waloddi Weibull et al. A statistical distribution function of wide applicability. *Journal of applied mechanics*, 18(3):293–297, 1951.

[93] Jianshu Weng and Bu-Sung Lee. Event detection in twitter. In *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*, 2011.

[94] Runquan Xie, Feida Zhu, Hui Ma, Wei Xie, and Chen Lin. Clear: A real-time online observatory for bursty and viral events. *PVLDB*, 7(13):1637–1640, 2014.

[95] Wei Xie, Cheng Li, Feida Zhu, Ee-Peng Lim, and Xueqing Gong. When a friend in twitter is a friend in life. In *Web Science 2012, WebSci '12, Evanston, IL, USA - June 22 - 24, 2012*, pages 344–347, 2012.

[96] Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. Topicsketch: Real-time bursty topic detection from twitter. In *ICDM*, pages 837–846, 2013.

[97] Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. Topicsketch: Real-time bursty topic detection from twitter. *IEEE Trans. Knowl. Data Eng.*, 28(8):2216–2229, 2016.

[98] Wei Xie, Feida Zhu, Siyuan Liu, and Ke Wang. Modelling cascades over time in microblogs. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, pages 677–686, 2015.

[99] Jaewon Yang and Jure Leskovec. Modeling information diffusion in implicit networks. In *ICDM*, pages 599–608, 2010.

[100] Yiming Yang, Thomas Pierce, and Jaime G. Carbonell. A study of retrospective and on-line event detection. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 28–36, 1998.

[101] Hongzhi Yin, Bin Cui, Hua Lu, Yuxin Huang, and Junjie Yao. A unified model for stable and temporal topic detection from social media data. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 661–672, 2013.

[102] Linyun Yu, Peng Cui, Fei Wang, Chaoming Song, and Shiqiang Yang. From micro to macro: Uncovering and predicting information cascading process with behavioral dynamics. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 559–568, 2015.

[103] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat-Thalmann. Who, where, when and what: discover spatio-temporal topics for twitter users. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 605–613, 2013.

[104] Tauhid Zaman, Emily B. Fox, and Eric T. Bradlow. A bayesian approach for predicting the popularity of tweets. *CoRR*, abs/1304.6777, 2013.

[105] Xin Zhang and Dennis E. Shasha. Better burst detection. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 146, 2006.

[106] Junzhou Zhao, John C. S. Lui, Donald F. Towsley, and Xiaohong Guan. Whom to follow: Efficient followee selection for cascading outbreak detection on online social networks. *Computer Networks*, 75:544–559, 2014.